Explorations in interactive illustrative rendering

Gerl, Moritz Alexander Christian

EXPLORATIONS IN INTERACTIVE ILLUSTRATIVE RENDERING

MORITZ GERL

The work presented in this thesis has been funded by a Ubbo Emmius scholarship and was carried out in the Institute for Mathematics and Computer Science according to the requirements of the Graduate School of Science (Faculty of Mathematics and Natural Sciences, University of Groningen). A collaboration with the Vienna University of Technology has been funded by the ViMaL project supported by the Austrian Science Fund (FWF), grant no. P21695.

Cover: Hatching illustration of the third lumbar vertebra.

Explorations in Interactive Illustrative Rendering
Moritz Gerl
Supervised by Dr. Tobias Isenberg
PhD thesis Rijksuniversiteit Groningen
ISBN 978-90-367-6063-8 (printed version)
ISBN 978-90-367-6062-1 (electronic version)

RIJKSUNIVERSITEIT GRONINGEN

EXPLORATIONS IN INTERACTIVE ILLUSTRATIVE RENDERING

Proefschrift

ter verkrijging van het doctoraat in de
Wiskunde en Natuurwetenschappen
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. E. Sterken,
in het openbaar te verdedigen op
vrijdag 15 maart 2013
om 11.00 uur

door

MORITZ ALEXANDER CHRISTIAN GERL

geboren op 18 april 1980
te Darmstadt, Duitsland

Promotor:	Prof. dr. J.B.T.M. Roerdink
Copromotor:	Dr. T. Isenberg
Beoordelingscommissie:	Prof. dr. O. Deussen
	Prof. dr. M. E. Gröller
	Prof. dr. B. Preim

It is truly useful
since it is beautiful.

Antoine de Saint-Exupéry

CONTENTS

1 INTRODUCTION

Illustrations are a powerful means of visually communicating information. The evolution and usage of graphical illustrations has a long history and illustrations are nowadays commonplace in a wide variety of domains. To name but a few, we can find illustrations in instruction manuals, in textbooks, and in scientific publications. To be able to create high-quality illustrations, however, one has to be artistically talented and undergo a lengthy training in fine arts and illustration to become a professional illustrator. Hiring such an illustration specialist is expensive and unfeasible in many cases. This often results in using self-made illustrations of insufficient quality. The problem is that the creation of high-quality illustrations is hard to accomplish for laypeople in illustration. This thesis presents two methods for interactive illustrative rendering that deal with this problem by making illustration expertise accessible to people without knowledge and skills in illustration. The proposed methods make the creation of illustrations available in domains where it has not been available before. Let us explain the benefit of the availability of illustration expertise to non-experts with an example.

Imagine a researcher in the need of a case-specific illustration which shall illustrate a novel finding in a scientific publication. The finding is of such a novelty that there is no existing illustration available to the researcher that illustrates the finding sufficiently well. The researcher cannot afford to hire a professional illustrator to create a suitable illustration. The researcher in our example must then resort to either using an illustration that does not exactly match the intended illustration purpose, to omit the illustration completely, or to create the illustration by him- or herself. All of these solutions will arguably have a negative impact on the quality of the scientific publication, unless the researcher is also a skilled illustrator. This problem can be alleviated by harnessing the benefits of methods for interactive illustrative rendering. These methods help to deal with the described problem by making illustration expertise available to non-experts in illustration. In this thesis, we present two methods that meet this demand. Besides the purpose of making illustration expertise available to non-experts, the methods presented in this thesis can as well be employed by professional illustrators to speed up the production process, to create accurate illustrations directly from 3D models or volume data, and to find new ways of visual expression. Let us briefly summarize the two methods that we present in this thesis.

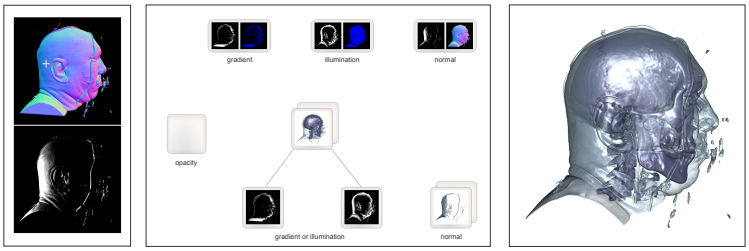


Figure 1.1: The interactive volume illustration system presented in Chapter 3. The system incorporates (left) an interface for brushing the contributions of input properties to rule-based visualization mappings as well as (center) a graphical rule specification interface to interactively control (right) the visualization result.

The first method presented in Chapter 3 is a graphical approach for the interactive creation of illustrative volume renderings (see Fig. 1.1). This method builds upon a framework that allows to specify the results of volume renderings with visualization rules [Rautek et al., 2007, 2008a]. We present a graphical user interface that makes the specification of such visualization rules more direct and intuitive than the original textual rule specification. Furthermore, we propose a graphical way of specifying the contributions of various input properties to the visualization mapping via brushing. Finally, we introduce a concept in Chapter 3 that facilitates to automatically augment arbitrary shader programs with rule-based rendering functionality. This flexible concept makes it possible to use arbitrary input and output properties in visualization rules and extends the range of achievable visualization mappings.

The second method presented in Chapter 5 is a method for interactively generating example-based hatching renderings of 3D models. A result of this pen-and-ink hatching method is shown in Fig. 1.2. The approach improves upon the synthetic and regular appearance of the results of previous approaches for computer-generated hatching. The improvement of the aesthetic quality and illustration effectiveness is achieved by the fusion of automatic by-example functionality with interactive editing functionality. The approach in Chapter 5 introduces a hierarchical style transfer model that allows us to learn the hatching style from hand-drawn pen-and-ink illustrations using image processing and machine learning. The style transfer model includes data representations that facilitate the interactive example-based synthesis of illustrations in the learned hatching style. The method in Chapter 5 comprises interaction capabilities that enable users to directly and intuitively adjust the hatching illustrations according to their aesthetic judgment and requirements. These interaction capabilities improve upon the aesthetic quality of the results that can be achieved with the method and facilitate an application of the hatching method for illustration and creative purposes.

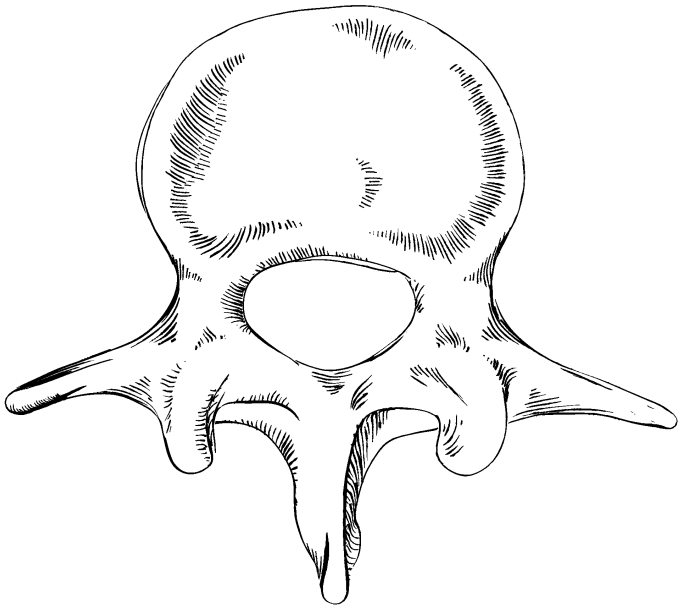


Figure 1.2: An illustration of a vertebra created with the hatching method in Chapter 5.

This thesis is structured as follows. In Chapter 2 we further motivate our work and draw a connection between the two distinct illustrative rendering methods that we present in this thesis. We illuminate the relation of the two approaches by looking at them in the light of the issue of implementing control over the result in illustrative rendering. We proceed with detailing our interactive volume illustration method in Chapter 3. In Chapter 4, we summarize our findings on control over the result that we learned from developing the volume rendering approach in Chapter 3. Based on these findings, we motivate the strategies for implementing control over the result that we devised for developing the hatching approach presented in Chapter 5. Next, we present the interactive example-based hatching method in detail in Chapter 5. Finally, we present ideas for future work and conclude the thesis in Chapter 6.

2

ILLUSTRATIVE RENDERING & CONTROL OVER THE RESULT

Large subdomains of computer graphics focus on the realistic depiction of virtual objects. Physical simulations of light and material coupled with optical models increasingly approach a photorealistic visual impression in depicting virtual scenes. Over the past three decades, rapid advances in computer graphics research have led to spectacular results in realistic rendering, a development that was at least partially fostered by the entertainment industry. Photorealism is, however, not the only depiction style that evolved in this process. The field of illustrative rendering or non-photorealistic rendering (NPR) emerged parallel to the described advances in realistic rendering. Non-photorealistic and illustrative rendering methods have also been applied to and developed for data visualization. This process formed the field of illustrative visualization [Viola et al., 2005; Bruckner, 2008; Rautek et al., 2008b].

2.1 NON-PHOTOREALISTIC RENDERING

The field of NPR is mainly concerned with the depiction of 2D images or 3D surface models in visual abstractions or styles that resemble traditional rendering media. A broad range of methods was developed that simulate rendering styles such as watercolor paintings, pencil drawings, or pen-and-ink illustrations [Gooch and Gooch, 2001; Strothotte and Schlechtweg, 2002; Kyprianidis et al., 2012; Rosin and Collomosse, 2012]. Some NPR methods are image filters, others are shading models, and others rely on the explicit calculation of drawing or painting marks, which often are strokes [Hertzmann, 2003]. In the majority of NPR methods the rendering result is algorithmically pre-determined. This means that the control over the appearance of the generated images lies solely in the hands of the *programmer*. This algorithmic control contrasts the artistic nature of the imagery generated by non-photorealistic and illustrative rendering methods. We will explain why this algorithmic control restricts the aesthetic quality and illustration effectiveness of illustrative rendering methods. In this thesis we examine two different strategies for implementing control over the result in illustrative rendering that improve upon a control that is solely in the hands of the programmer: user interaction and learning from examples.

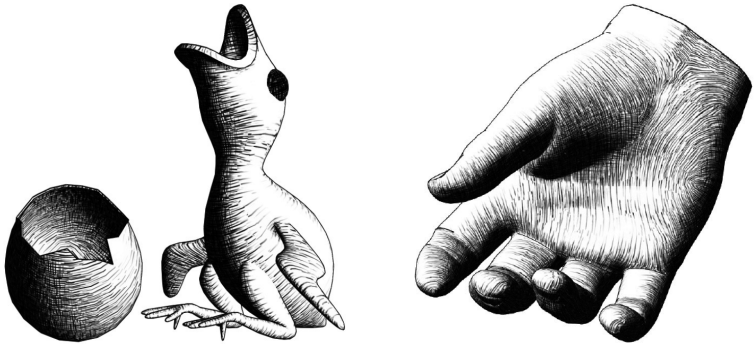


Figure 2.1: Fully automatic real-time hatching by Praun et al. [2001].

When we speak of *control* and *controlling* in this thesis, we speak of *control* in the sense of *command*, *specification*, *regulation*, and *guidance*. We do not refer to *control* and *controlling* in the sense of *checking*, *testing*, and *verifying*. Furthermore, we use the term *rendering function* in this thesis as a generic term for a function that defines a mapping from a set of input arguments to a set of rendering outputs. An input argument of such a generic *rendering function* can be any input value on which the visual mapping is based (e. g., a color value of an input raster image, a geometric feature of a 3D surface, or a data feature in a volume dataset). An *input argument* can also be a rendering parameter (e. g., a parameter that controls the brightness of the rendering). The *rendering output* can be simply a color and opacity value per pixel, or a value of a parameterized visual abstraction (e. g., the local density of hatching strokes). In the parts of this thesis that deal with volume rendering, we use the term *visualization mapping* accordingly to the definition of the term *rendering function* given here. Having introduced this terminology, let us proceed with examining the different strategies for controlling the result in illustrative rendering.

Many of the established NPR methods are fully automatic. They employ a fixed rendering pipeline and offer users a *'one-button-solution'* (apart from the necessity to manually preprocess the model to be rendered). In such fully automatic methods, the result is controlled solely by the rendering algorithm, which implements a *static rendering function*. This fully automated approach is very convenient and time-effective but lacks the creative freedom users might be expecting, in particular if users stem from an artistic background. The need for *'putting the artist in the loop'* was advocated by Seims [1999] and Salesin [2002] already in the early days of NPR, but seems to not have attracted the full awareness of the NPR community. To give an example, the texture-based hatching method

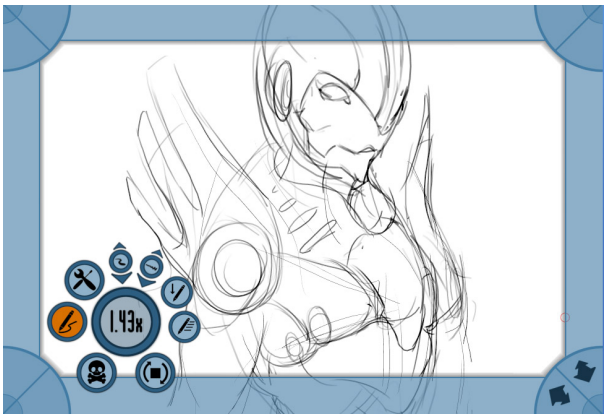


Figure 2.2: Fully interactive freehand sketching system by [Nijboer et al., 2010].

presented by Praun et al. [2001] allows users to create attractive hatching renderings of 3D models in real time (see Fig. 2.1). The method, however, does not allow users to change the appearance of the resulting drawings. Users of the hatching renderer by Praun et al. [2001] cannot, e. g., manually darken or brighten a particular region. This lack of possibility for user interaction limits the creative freedom of the hatching method and, thus, the possibilities of a deployment of the method in creative environments such as the entertainment industry. On the one hand, the fact that the user does *not have to* perform such interactions to achieve aesthetically pleasing results surely is a benefit of the technique by Praun et al. [2001]. On the other hand, we see it as a limitation that the user *cannot* freely and directly adjust the rendering to his or her needs.

Let us treat the different approaches to controlling the result in illustrative rendering as a continuum from fully automatic to fully interactive. In this continuum, fully interactive drawing systems form the other extreme as compared to fully automatic NPR systems such as the hatching renderer by Praun et al. [2001]. An example for a fully interactive drawing system is the freehand sketching system presented by Nijboer et al. [2010]. Fig. 2.2 shows an example screenshot of the freehand sketching system by Nijboer et al. [2010]. In this system, users can draw strokes, interact with already drawn strokes, as well as interact with the canvas using context-sensitive gestures. Users of such a drawing system have full creative freedom and control over the result image but are required to draw the entire image 'from scratch,' i. e., without the assistance of a 3D model or a rendering algorithm. The described interaction continuum is illustrated in Fig. 2.3. We continue with examining some strategies for implementing control over the result that are situated between the two extremes of fully automatic and fully interactive.

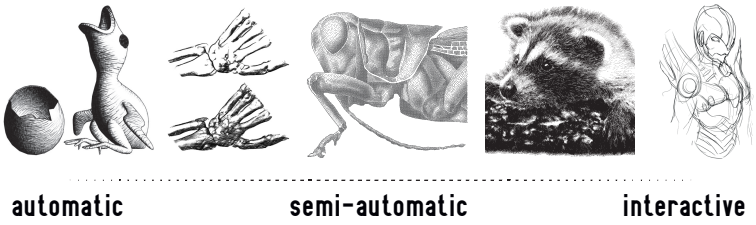


Figure 2.3: Interaction continuum. The various strategies for implementing control over the result in NPR span a continuum that ranges from fully automatic to fully interactive. The image to the left is generated fully automatically while the image to the right is drawn by hand. The images from left to right are results of Praun et al. [2001], Gerl [2006], Deussen et al. [2000], Salisbury et al. [1997], and Nijboer et al. [2010].

Many NPR methods provide the possibility to interact with the result by the means of interacting with rendering parameters. For stroke-based methods, these are parameters such as the density or the width of the strokes. Users of common NPR systems can adjust the result by entering numeric values or adjusting sliders to control such parameters. The automatic result is updated after a rendering parameter has been changed by the user. The image synthesis is still fully automatic in such a system, while the possibility to specify rendering parameters gives some control over the result to the user. In our interaction continuum in Fig. 2.3, such an interaction via rendering parameters is the first step away from a fully automatic system towards an interactive system. Interaction by parameter adjustment is an effective means of interaction in many cases, in a sense that one can quickly change the visual appearance of the result without much effort. Parameter tweaking is, however, an indirect means of adjustment and provides only limited control over the result. It is indirect because it does not allow users to directly adjust the drawing elements, but rather interact with the parameters that influence the drawing elements. The control over the result of NPR systems provided by parameter adjustment is limited because of two major reasons, which are the common restriction to global adjustments and the restrictions imposed by a pre-determined range of achievable visual effects.

First, the control given by parameter adjustment is limited because only global parameters are provided in many NPR systems. For example, a stippling renderer might permit users to adjust the size of *all* the stippling marks in the result, but not exclusively the size of stipples within a particular region. To illustrate this global behavior, Fig. 2.4 shows the adjustment of a global brightness parameter in a hatching renderer for volume data by Gerl [2006]. Here, adjusting the brightness parameter brightens or darkens the entire rendering. The global adjustment, however, does not allow to selectively darken or brighten particular regions.

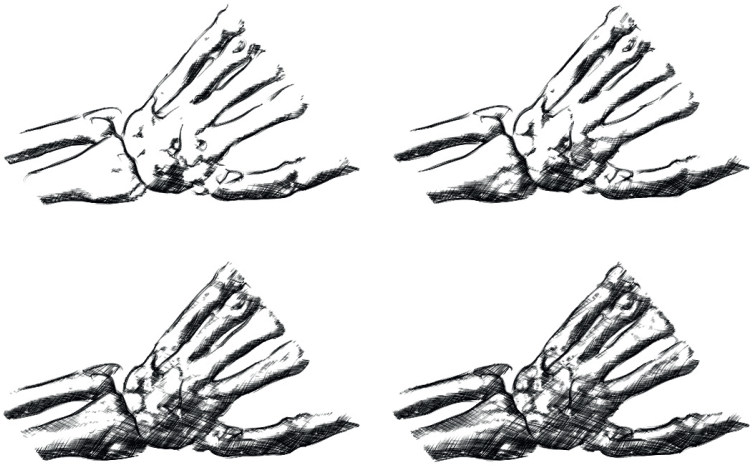


Figure 2.4: Global brightness parameter adjustment in volume hatching [Gerl, 2006].

Second, the control given by parameter tweaking is limited because it restricts the possible interactions to a particular range of effects which is pre-determined by the set of provided parameters. The range of achievable effects is restricted by the complexity of the parameter space. In our stippling renderer example, users might be able to adjust the size of the stippling marks, the stippling density, and a perturbation factor that adds a random offset to the stipple positions. The range of adjustments that are possible to perform with this example stippling renderer is limited to the effects that can be achieved by modifying the three described parameters.

A brushing interaction for controlling a stippling renderer such as presented by Deussen et al. [2000], or for controlling a hatching renderer as introduced by Salisbury et al. [1994, 1997], in contrast, breaks free from this restriction to global adjustments and the limited range of achievable effects. It allows users to directly interact with the result, to make local adjustments, and to modify the result independently of a predetermined range of visual effects that can be achieved with parameter tuning. This increased range of achievable visual effects combined with the creative freedom given by the possibility for interaction results in computer-generated illustrations that look less mechanical and more hand-drawn than fully automatic methods, as shown in the example in Fig. 2.5. Such semi-automatic illustration systems are located in the center of our described interaction continuum, as illustrated in Fig. 2.3.

Interactive NPR systems permit human intervention and provide the user with control over the result. This control opens up the possibility to enhance the expressiveness, the hand-drawn appearance, or the 'charac-

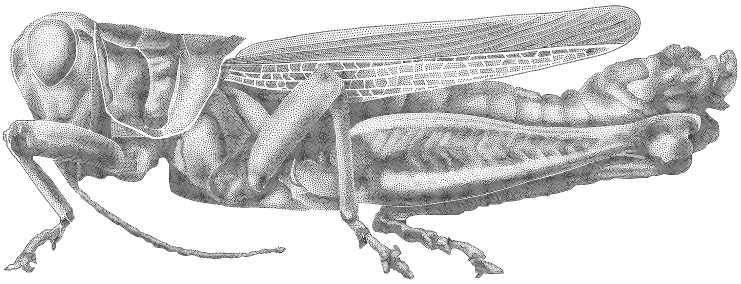
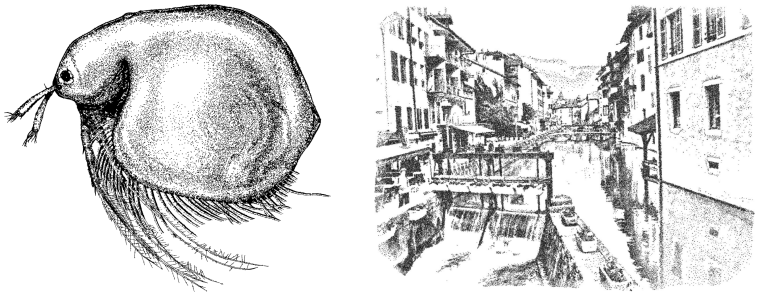


Figure 2.5: Locally adjustable stippling by Deussen et al. [2000].

ter' of the resulting images. Another means of achieving this goal is to learn the parameters and/or the properties of a drawing style from hand-drawn examples. This brings our focus back to automatic systems. First of all, many traditional NPR systems implement an automatic control over the result via relying on a rather simple mapping from a small set of image or surface features to rendering parameters. The *rendering functions* used by early NPR systems are rather simple mappings. Such a simple mapping is, e. g., to map the lighting intensity to the stippling density in the generation of stippling images from 3D models or, respectively, to map the image intensity to the stippling density in the generation of stippling drawings from 2D images. These mappings roughly approximate the tone and shading that can be found in many hand-drawn stippling images. However, the described simple mappings do not suffice to faithfully reproduce all the stylistic variations and tonal detail present in hand-drawn stipplings. Recent efforts in NPR address the described limitation of a simplistic mapping by learning a model of the drawing style present in hand-drawn examples and using the learned model to generate example-based renderings. For example, Kim et al. [2009] learn a statistical model of stipple distributions and stipple textures to transfer the stippling characteristics from hand-drawn examples to computer-generated stippling renderings of 2D input images. Related to this approach, Martín et al. [2011] model the stippling process as a halftoning procedure and transfer individual stippling marks from an example illustration to a target illustration based on local image characteristics. This example-based approach to controlling the rendering result opens up the possibility to incorporate a model of human virtuosity in the rendering process. This strategy results in computer-generated illustrations that exhibit more of a 'character' than the results of conventional methods, which rely on simple mappings of image features to rendering parameters. This improvement might become apparent in the examples shown in Fig. 2.6. The mentioned techniques for example-based stippling have in common that they both establish models of the characteristics and usage of drawing elements or



(a) (b)

Figure 2.6: Stippling by example by (a) Kim et al. [2009] and (b) Martín et al. [2011].

drawing marks (stipples). The described systems for stippling by example operate directly on the rendering primitives.

Instead of learning the statistical properties of the drawing elements themselves, another means of example-based illustrative rendering is to learn the parameters of a particular rendering style as proposed by Hamel and Strothotte [1999]. This learning of rendering parameters is linked to the previously described interactive navigation of parameter spaces to adjust the result of a non-photorealistic or illustrative rendering system. In the formerly described interaction case, the parameter space is navigated (inter)actively by the user. In the learning case of Hamel and Strothotte [1999], the parameter space is navigated automatically by the machine. The work presented in this thesis explores the interactive, the semi-automatic, as well as the automatic navigation of parameter spaces for illustrative rendering in two specific case studies. The goal of this effort is to improve upon the aesthetic and illustrative quality as well as the interaction possibilities of computer-generated illustration. A particular contribution of this thesis is the combination of machine learning and user interaction to implement an effective semi-automatic control over the result of illustrative rendering methods. This combination forms an interactive example-based rendering system. In this semi-automatic approach, the parameter space is navigated partially by the machine and partially by the user. The described combination is related to the semi-automatic visual analysis of multi-dimensional data proposed by Fuchs et al. [2009], who describe a visual human+machine feedback learning system for hypotheses generation. In this system, Fuchs et al. [2009] combine human and automatic reasoning for the analysis of multi-dimensional data. This approach relates to our efforts of combining automatic and interactive control over multiple inputs and outputs of rendering functions. Let us proceed with outlining the issue of control over the result with respect to illustrative data visualization.

2.2 ILLUSTRATIVE VISUALIZATION

The problem of parameter space navigation becomes even more challenging when we leave the world of illustrative rendering of 2D images and 3D surface models and enter the world of illustrative volume rendering. Ebert and Rheingans [2000] were among the first to apply non-photorealistic rendering methods to direct volume rendering in order to create illustrative visualizations of volume data. Their seminal work also appeared as Rheingans and Ebert [2001]. The volume stippling renderer proposed by Lu et al. [2002] is an early work towards the simulation of traditional rendering media in direct volume rendering. In the following years, a range of techniques for illustrative visualization evolved. The interested reader might consult the surveys by Viola et al. [2005], Bruckner [2008], and Rautek et al. [2008b]. In the generation of illustrative visualizations, both the parameters of the NPR methods and the parameters of the visualization methods have to be taken into account. This makes the control over the result more difficult than for rendering 2D images or 3D surface models. Even without respect to the usually numerous NPR parameters, the parameter space for visualizing volume data is intrinsically more difficult and unintuitive to navigate than the parameter space for visualizing 3D surface models. This is due to the fact that in volume rendering one has to specify which structures within the volume ought to be visualized. This specification is usually not necessary in surface rendering, where most commonly the entire surface is depicted.

The complexity of specifying the visibility of structures within a volume dataset is directly related to the dimensionality of the data, but already challenging enough for 3D scalar fields. Transfer functions are the standard way of performing this specification, and surely are a powerful and effective tool. Despite of their effectivity, transfer functions are subject to some limitations. First, the usage of transfer functions requires a basic knowledge of the visualization process and a fair amount of training to be used efficiently. Second, transfer functions are an indirect means of specification in a sense that users do not interact directly with the result but with an abstract data representation in a separate region of the screen. Third, transfer functions have a limited range of possible mappings. Some mappings are impossible to realize using standard transfer functions, e. g., visualizing the skull-surrounded brain in an MRI scan of a head. As the density of the bone tissue is higher than the density of the brain matter, the brain would always be occluded by the skull. This circumstance requires the use of advanced transfer function approaches. With growing complexity of the approaches and with the increasing complexity of the involved parameter spaces, however, advanced transfer function methods run the risk of being counterintuitive to use and demand more and more user expertise and training.

Apart from the usability aspect, the effects that are achievable with transfer functions are limited by the domain and co-domain on which the transfer function operates. In the simplest case, a transfer function only allows users to map the original *data values* to *opacity* and *color* values. A mapping from, e. g., *'data gradient'* to a value of a parameterized visual abstraction (e. g., *'cartoon shading'*) again requires advanced transfer function methods. This limited range of visual effects that are achievable with transfer functions is similar to the restrictions inherent to NPR methods that employ simple mappings (e. g., that map image intensity to stippling density) as described in Section 2.1. The problem is similar: the range of visual effects that can be achieved with the rendering function that models a visual abstraction in a rendering algorithm is limited by the number of inputs and outputs of the rendering function. Consequently, the accuracy in which a traditional depiction style can be approximated or simulated by an illustrative rendering algorithm is limited by this very restriction of achievable visual effects. Increasing the number of inputs and outputs of the rendering function unfortunately also increases the problems of specifying an adequate mapping. One approach to this problem is to employ machine learning to learn an adequate mapping. A different approach to this problem is to apply sophisticated means of user interaction, as discussed in Section 2.1. Note that data derivatives such as the gradient are sensitive to noise and are usually not standardized. Due to this, transfer functions based on data derivatives up to now have had only little importance for the clinical practice in medical visualization. Nonetheless, advanced interaction methods might help to better harness the potential of advanced transfer functions concepts.

The notion of semantics-driven volume rendering that was introduced by Rautek et al. [2007, 2008a] deals with the described problems of control over the result and parameter specification in illustrative visualization. The framework of Rautek et al. [2007, 2008a] allows users to specify a mapping from data properties to visual attributes using natural language. Instead of editing a transfer function to specify a visualization mapping, users of a semantics-driven visualization system formulate the desired mapping as a textual rule, for example *'if density is high then color is green.'* This rule-based approach alleviates many of the described problems of transfer function design and is a powerful tool for creating illustrative visualizations, as can be seen from the examples of Rautek et al. [2008a] that are shown in Fig. 2.7. Formulating visualization rules in natural language is, however, subject to new limitations. One part of this thesis deals with these new limitations inherent to a specification of the rendering function using natural language.

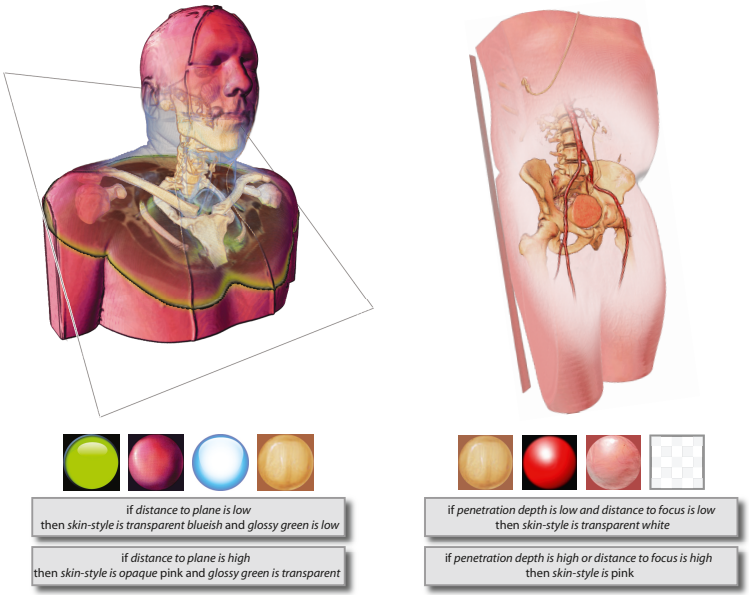


Figure 2.7: Semantics-driven illustrative visualizations by Rautek et al. [2008a].

2.3 INTERACTIVE SEMANTICS-DRIVEN VOLUME RENDERING

The first core part of this thesis in Chapter 3 is devoted to improving upon the limitations of a textual rule formulation for semantics-driven visualization as well as to the problem of inputs and outputs that can be used in a visualization mapping. The formulation of visualization rules in natural language is a powerful and effective way of specifying a visualization mapping. It abstracts from the concept of transfer functions and provides users with a natural way of controlling the resulting visualization. A textual rule formulation has the downside, however, that the user has to mentally envision the effects of possible rules or modification of rules before the rule is created or modified. This limits the possibilities and the efficiency of exploring different mappings. In our framework for interactive semantics-driven volume rendering (Chapter 3), which builds upon the foundation of Rautek et al.'s [2007; 2008a] work, we propose ways of dealing with this limitation. In Chapter 3 we describe a graphical user interface for rule specification (see Fig. 1.1). The interface provides the user with visual feedback in the rule specification process in the form of previews of possible results. The graphical control and visual feedback assist users in the rule specification and provide new exploration capabilities. Fig. 2.8 shows a result that was generated with the semantics-driven volume rendering system presented in Chapter 3.

(a) (b) (c)

Figure 2.8: A result of the interactive volume illustration system presented in Chapter 3. The system allows users to apply (a), (b) visual abstractions to the result via brushing to interactively generate (c) rule-based illustrative visualizations.

A second limitation of the original semantic layers technique by Rautek et al. [2007, 2008a] is the fact that the *data semantics* are specified in a function editor. The term *data semantics* refers to a function of the input values in the visualization mapping such as, e. g., the phrase *'density is high'* in the rule *'if density is high then color is green.'* This indirect specification of rendering parameters contrasts the otherwise direct specification offered by semantics-driven volume rendering. In Chapter 3 we propose a technique to specify data semantics by analogy via brushing on visualizations of semantic data properties. The user is provided with a rendering of, e. g., the data property *density* and specifies which data ranges ought to be visualized by brushing on the rendering. This visual approach again makes use of visual feedback and allows to specify data semantics more directly and intuitively than using a function editor. The brushing of data semantics complements the graphical rule specification. Together, these two graphical approaches give the user direct and interactive control over the result.

Another aspect of the framework described in Chapter 3 deals with the previously described problem of inputs and outputs that can be used in visualization mappings. In Chapter 3, we propose a technique that facilitates the usage of arbitrary inputs and outputs in visualization mappings. The core concept of this technique is to automatically add rule-based rendering functionality to arbitrary shader programs. This concept of semantic shader augmentation makes it possible to use the visual abstraction created by shader programs as the output of visualization rules. This means that users of the presented system can use the output of a shader program, e. g., *'cartoon shading'* as the output of a visualization rule. Furthermore, the concept permits to use arbitrary variables in the shader program as input to visualization rules. The visualization mapping

can thus be based on any variable in the shader program, e. g., *density, gradient, normal, curvature, lighting, etc.*. This increases the flexibility of semantics-driven visualization and the range of possible mappings. The described concept of semantic shader augmentation allows the user to base the rendering on a multitude of data features. The user of the described system can control the result by selecting data features of interest and by graphically specifying rules and data semantics. This gives the user a very flexible control over the visualization result. The downside of this flexibility is that it *requires* the user to make a selection of data features and to use the selected features in visualization rules in a meaningful way. In Chapter 5, we explore a different approach to implementing control over the result which is less flexible, but in turn provides more assistance to the user.

2.4 INTERACTIVE EXAMPLE-BASED HATCHING

The second core part of this thesis in Chapter 5 presents a different way of using data features as input rendering parameters. Instead of actively selecting semantic data properties and interactively specifying their contributions to the visualization mapping, we here apply machine learning to automatically learn functions that map data (resp. surface) properties to output rendering parameters. Interaction is then provided on a high level, while the learned functions form the low-level basis for the visual appearance that is achieved with the rendering style.

In Chapter 5 we discuss a technique for the interactive example-based generation of pen-and-ink hatching illustrations from 3D polygonal models. The main goal of this technique is to improve upon the aesthetic quality of the results of existing methods for computer-generated hatching. We aim at enhancing the overly regular and rather synthetic appearance of the results of existing methods by learning the drawing style from hand-drawn examples as well as by providing possibilities to interact with the resulting hatching illustration. Results of the interactive example-based hatching method are shown in Fig. 1.2 and Fig. 2.9.

One contribution of the hatching method described in Chapter 5 is that the drawing style is learned from hand-drawn and scanned-in example illustrations. This learning from real-world hatching illustrations is facilitated by employing image processing to establish an analytical description of the hatching strokes in the example image and by using a 3D scene whose projection closely matches the example illustration. Based on this setup, we use machine learning to establish a model of the drawing style in the example illustration. We then apply this learned model to target 3D meshes in order to synthesize example-based hatching renderings of the target meshes.

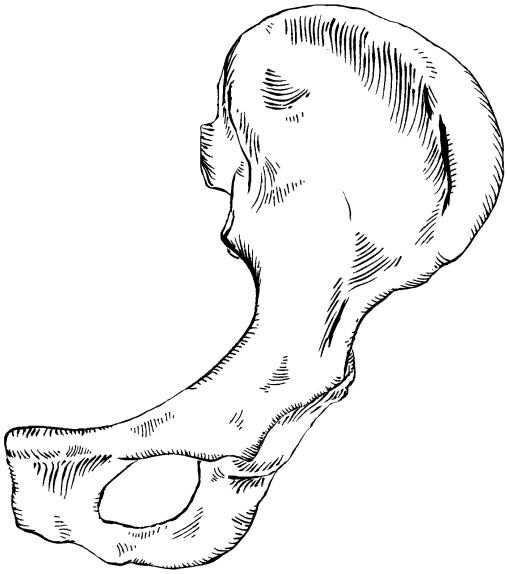


Figure 2.9: A result of the interactive example-based hatching system presented in Chapter 5. The rendering shows a pen-and-ink illustration of a hip bone.

We use a set of surface features such as *shading, facing ratio, curvature, image-space coordinates, etc.* and train functions that map these features to rendering parameters. We employ three different types of functions to learn the characteristics of hatching regions, directions, and distances. Applying these functions in the synthesis stage yields hatching regions and strokes that incorporate the learned drawing characteristics. We thus automatically learn a mapping of data features to rendering parameters, as opposed to the interactive mapping specification chosen in the volume rendering approach presented in Chapter 3.

In the synthesis stage, our approach for hatching by example automatically infers stroke regions, directions, and distances from surface features. We provide a set of high-level user interactions that complements and improves upon this automatic generation. These interactions allow users to adjust the rendering to their requirements and aesthetic judgment. Two of these user interactions use a brushing metaphor and improve upon the disadvantages of a numerical global parameter tuning that we described in Section 2.1. One of the interactions is a brushing tool that permits users to edit a surface direction field which serves as a reference field for inferring stroke directions. This direction field brushing allows users to flexibly edit the trajectories of hatching strokes. Another interaction permits users to brush with patches of hatching strokes. This interaction gives a direct and intuitive control over the regions in which hatching strokes are generated.

The described semi-automatic solution combines the advantages of an automated control over the result with the flexibility and creative freedom gained by giving control over the result to the user. The advantage of the automated control is the example-based generation of hatching strokes that describe the target surface. The advantage of the interactive control is the possibility to freely and directly specify where these hatching strokes are generated, as well as the possibility to interactively and locally change the properties of the generated strokes. The combination of automatic learning and interactive adjustment of the hatching illustrations results in illustrations that arguably exhibit more 'character' and a more hand-drawn appearance than the results of existing methods which are either not example-based or not interactive. This improvement of the 'character' of the results is due to the fact that the results are influenced by human virtuosity through both the example-based automatic and the interactive components of the hatching system. We distribute the control over the result in a manner that allows us to improve upon the aesthetic quality of the hatching illustrations. Let us briefly summarize our findings on illustrative rendering and control over the result.

2.5 SUMMARY

Implementing an appropriate control over the result in non-photorealistic and illustrative rendering is a challenging task. Numerous approaches have been proposed in the literature to deal with this task. These approaches range from fully automatic methods, over methods relying on global rendering parameters, over semi-automatic approaches, to fully interactive approaches. In illustrative volume visualization, the control over the result is particularly challenging because it involves the additional burden of specifying which structures in a volume dataset ought to be visualized. The parameter space is here intrinsically more complex and unintuitive to navigate. In this thesis, we describe two different methods for illustrative rendering that explore two different strategies for controlling the result. The first method detailed in Chapter 3 is concerned with interactive semantics-driven volume rendering. It evolves around a flexible definition of the input and output that can be used in the visualization mapping as well as around a graphical user interface to directly and interactively control the desired rule-based mapping assisted by visual feedback. The second method described in Chapter 5 is an interactive approach to the example-based generation of pen-and-ink hatching illustrations of 3D meshes. Here, a mapping from input to output parameters of the rendering function (i. e., surface features to hatching properties) is learned automatically and can be applied interactively by the user to create the desired hatching illustrations. This semi-automatic approach

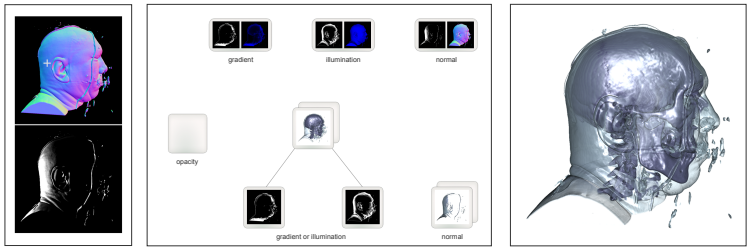
to implementing control over the result allows us to combine the benefits of a fully automatic illustration system with human virtuosity. The list below summarizes our observations on implementing control over the result in illustrative rendering.

OBSERVATIONS

- automatic control over the result allows for a rapid generation of results but lacks creative freedom;
- interactive control over the result permits the user to enhance the aesthetic quality of the results;
- control over the result is particularly challenging in illustrative visualization due to the increased complexity of the parameter space;
- rule-based methods are a way to implement control over the result in illustrative rendering;
- rule-based approaches can be enhanced by graphical user interfaces;
- example-based methods are a way to implement advanced automated control over the result in illustrative rendering; and
- interactive example-based approaches allow for a twofold influence of human virtuosity on the result (both algorithmic and user influence)

3

INTERACTIVE SEMANTICS-DRIVEN VOLUME RENDERING



ABSTRACT: *In this chapter, we discuss an interactive graphical approach for the explicit specification of semantics for illustrative volume visualization. This explicit and graphical specification of semantics for volumetric features allows us to visually assign meaning to both input and output parameters of the visualization mapping. This is in contrast to the implicit way of specifying semantics using transfer functions. In particular, we demonstrate how to realize a dynamic specification of semantics which allows to flexibly explore a wide range of mappings. Our approach is based on three concepts. First, we use semantic shader augmentation to automatically add rule-based rendering functionality to static visualization mappings in a shader program, while preserving the visual abstraction that the initial shader encodes. With this technique we extend recent developments that define a mapping between data attributes and visual attributes with rules, which are evaluated using fuzzy logic. Second, we let users define the semantics by analogy through brushing on renderings of the data attributes of interest. Third, the rules are specified graphically in an interface that provides visual clues for potential modifications. Together, the presented methods offer a high degree of freedom in the specification and exploration of rule-based mappings and avoid the limitations of a linguistic rule formulation.*

3.1 INTRODUCTION

Letting users specify meaningful mappings from multiple volumetric data attributes to visual attributes is a challenging problem in direct volume rendering. A common approach is to design multi-dimensional transfer functions, which are flexible but also complex and demanding to use. The design of multi-dimensional transfer functions requires expert knowledge and is often provided in suboptimal interfaces. To address this issue, alternative ways of defining the visual-

ization mapping have been investigated. For instance, Rautek et al. [2007, 2008a] present a semantics-driven visualization framework that enables users to specify a mapping from multiple data features to visual attributes as textually formulated rules. This method allows them to explicitly define semantics for attributes of interest. Their approach to parameter specification for volume rendering bears great potential for creating illustrative visualizations. For example, the data attribute *high density* can be mapped to the visual attribute *cartoonish shading* by formulating an according rule. Formulating the visualization rules textually, however, is rather rigid and provides few possibilities for exploring different mappings. Textual rule specification is limited by the ability of the user to mentally envision and assess the effects of potential rules beforehand, and with that devise the appropriate rules. Thus, exploration is rather limited, tedious, and time-consuming. Furthermore, using a function editor to define data semantics (e. g., *high density*) is a rather indirect way of parameter specification. This contrasts the direct specification offered by the semantic-layers concept.

Let us briefly explain the usage of the terms *'semantic'* and *'semantics'* in this thesis. We use these terms in a restricted meaning that does not exactly comply with the common usage of the terms in other areas of computer science. We use *'semantic'* in the sense of *'to attribute meaning to a data property.'* In the example in the previous paragraph, the data property *density* is attributed meaning by being used in a visualization rule. It is attributed meaning in the sense that the data property is assigned a role in the visualization process and thus influences the visualization result. We use the term *'semantics'* accordingly.

To address the issues described above, we present a semantics-driven visualization technique that combines the advantages of a graphical specification of mappings with the illustration capabilities provided by a rule-based approach. The presented technique incorporates a new way of dynamically specifying the input and output of the visualization mapping, allows to define the mapping by analogy, and makes use of a graphical user interface for specifying visualization rules. This provides a more direct control of semantics and allows us to overcome the restrictions of a linguistic rule specification. Our goal is to provide a flexible tool for the specification and dynamic exploration of meaningful visualization mappings. We employ the following methods to achieve this goal:

Semantic shader augmentation: We present a technique for automatically augmenting an arbitrary shader program with semantics-driven rendering functionality. This technique replaces the visualization mapping present in an input shader program with a rule-based rendering method, while preserving the visual abstraction that the input shader originally generates. This semantic shader augmentation enables a flexi-

ble and dynamic definition of the input and output of the visualization mapping and extends the range of possible mappings. It permits users to rapidly create semantics-driven visualizations based on arbitrary variables in an arbitrary input shader program.

SEMANTICS BY ANALOGY: We let users define the visualization mapping by analogy. Specifically, users define contributions of properties to the mapping through brushing on visualizations depicting properties of interest. The final result images are then visually analogous to the brushed data ranges. This brushing on visual data representations permits a direct, flexible, and interactive definition of a semantic mapping. Furthermore, it allows users to intuitively define semantic mappings of two- and three-dimensional data properties, which was not feasible in previous approaches. The brushing interaction improves the usability and directness of specifying semantics-driven mappings.

GRAPHICAL RULE SPECIFICATION: We present a graphical interface for specifying visualization rules. With this interface, rules can be specified and modified by interacting with dedicated widgets. These widgets provide visual feedback on the semantic entities they represent, which allows to visually assess the effect of rules, or modifications to rules. The interface, therefore, makes use of people's visual information-processing capabilities in the rule specification process and opens up new possibilities for the exploration of semantics-driven mappings. It, thus, reduces the restrictions of a textual rule formulation.

Together, the described concepts form a technique for the interactive exploration of semantics-based visualizations (Fig. 3.3). Our technique is particularly well-suited for illustration purposes and can be employed as a tool for domain experts to quickly create illustrations from volume data. For instance, in the visualization of medical volume data it allows users to generate case-specific illustrations with respect to the diagnostic purpose or the treatment to be illustrated. The exploration capabilities of our technique may also be well-suited for educational applications. Furthermore, our interactive graphical approach to parameter specification is a step towards an illustrative volume rendering system which is suited for being used by scientific or medical illustrators.

We first outline related work in Section 3.2. Next, in Section 3.3 we describe our framework for semantics by analogy. In Section 3.4 we show and discuss exemplary results that are generated with our technique. In Section 3.5 we report on an evaluation of our technique. We then discuss its limitations in Section 3.6. Finally, we conclude the chapter and describe possibilities for future work in Section 3.7.

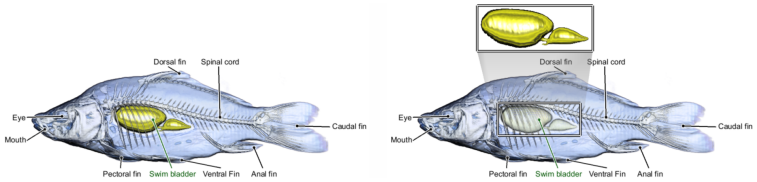


Figure 3.1: A direct volume illustration system by Bruckner and Gröller [2005].

3.2 RELATED WORK

The work presented in this chapter relates to research in rule-based visualization, selective application of visualization styles, multi-variate data visualization, graphical user interfaces for specifying visualization parameters, as well as to volume rendering using dynamic shader generation.

Early work on automated generation of illustrations based on rules was done by Seligmann and Feiner [1991]. They present a system for the generation of intent-based illustrations using design rules. The text-to-scene method introduced by Coyne and Sproat [2001] follows a similar approach. It enables the translation of simple semantics to images. Instead of a textual rule specification, we propose a graphical one in this chapter. Along these lines, Svakhine et al. [2005] use illustration motifs to gear visualizations towards the intended audience. Another semantics-based graphical interface for the specification of a multi-dimensional mapping is presented by Rezk-Salama et al. [2006]. Our method differs from these approaches by introducing dynamically generated semantics into the mapping process and by enabling users to specify a mapping based on direct data representations.

The methods that we introduce in this chapter build, in particular, upon techniques presented by Rautek et al. [2007, 2008a]. They introduce semantic layers and interaction-dependent semantics which allow to interactively create illustrative visualizations based on textual rules. This is realized with the help of fuzzy logic. Attributes of interest are interpreted as fuzzy sets, whose membership functions describe the attributes' contributions to the mapping. Fuzzy-logic arithmetics is applied to evaluate the illustration rules. In this work, we propose methods that extend the flexibility and exploration capabilities offered by Rautek et al.'s system using a graphical and analogy-based approach.

The application of different visualization techniques for selected subsets of a volume was proposed by Hauser et al. [2001] with two-level volume rendering. Other work in this direction [Csébfalvi et al., 2001; Lum and Ma, 2004; Bruckner and Gröller, 2005; Tietjen et al., 2008] further examines the selective application of styles and rendering attributes (see Fig. 3.1). Instead of an a-priori selection of volume subsets to map

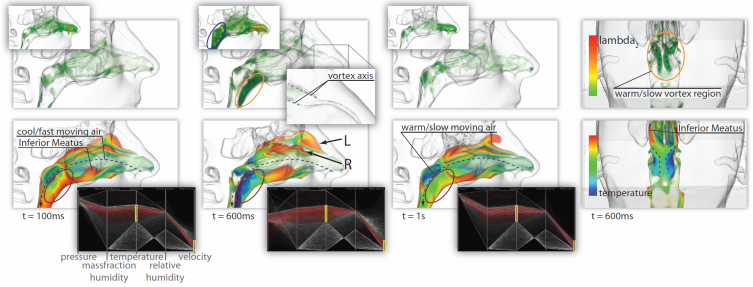


Figure 3.2: Visual exploration of nasal airflow by Zachow et al. [2009].

to different visualization techniques, we present a method for interactively exploring the selective application of visual attributes based on semantics-driven visualization rules.

Providing the user with exploration possibilities was successfully applied by Gasteiger et al. [2011] and by Neugebauer et al. [2011] for the exploration of blood flow in cerebral aneurysms. Both of these approaches use focus+context visualizations, which can also be achieved with the approach presented in this chapter. The approaches of Gasteiger et al. [2011] and of Neugebauer et al. [2011] provide exploration capabilities in order to assist users in insight generation and decision making processes. Our exploration capabilities, in contrast, focus on the exploration of different visual mappings for illustration purposes.

In this work we discuss a technique for specifying a mapping from a multi-variate data space to visual attributes. This relates to the notion of multi-dimensional transfer functions introduced by Kniss et al. [2002]. We deviate from this concept by basing the mapping on explicitly defined semantics and by providing tools to define the behavior of these semantics interactively. Other approaches also rely on explicitly defined semantics. Both McCormick et al. [2004] and Stockinger et al. [2005] use mathematical expressions to formulate the visualization mapping. Woodring and Shen [2006] use set operators and numerical operators for the comparative visualization of multi-variate and time-variate data. Another rule-based system is presented by Sato et al. [2000] who use rules to classify tissue structures in multi-modal datasets. We believe that rule-based visualization bears great potential for illustrative visualization and the realization of semantic mappings, but identify formal and textual rule formulations as rather rigid and non-exploratory. For this reason, we combine rule-based rendering with the benefits of graphical user interfaces for defining multi-dimensional mappings. Tzeng et al. [2005] present such an interface that permits the user to specify the input to a multi-dimensional data classifier via brushing. Fuchs et al. [2009] also exploit the direct editing and visual feedback capabilities of data brushing.

They propose a framework for semi-automatically deriving hypotheses on multi-variate data with the help of visual human and machine learning. Another interface for exploring mappings of multi-dimensional data is presented by Zachow et al. [2009]. They allow to brush values of interest in multiple linked abstract data representations to visually explore nasal airflow (see Fig. 3.2). All these systems have in common that the mapping is specified in an abstract data space. In contrast to this, our method allows a mapping specification with brushing on a more direct data representation, through visualizing attributes of interest and permitting the user of our system to brush on these visualizations. Apart from this, we also employ design galleries for parameter specification, as introduced by Marks et al. [1997].

The technique presented here makes use of automatic shader generation methods. In this context, Rössler et al. [2008] propose a technique for dynamically generating shader code for multi-volume raycasting from a graphically defined abstract render graph. Similar to this approach, we exploit the flexibility and abstraction from GPU programming offered by dynamic shader generation. We differ from Rössler et al. [2008] in realizing a rule-based approach and in the user interface for specifying visualization parameters.

3.3 A FRAMEWORK FOR SEMANTICS BY ANALOGY

The overall concept of our framework for semantics by analogy is depicted schematically in Fig. 3.3. It consists of three components that influence the resulting interactive illustration. The input to our system is an initial shader program selected by the user. The *input shader program* is a combination of GLSL and special tags that surround variables of potential interest. The input shader is parsed and *automatically* augmented with rule-based rendering functionality by a pre-compiler (Section 3.3.1). This results in an *augmented shader program* as output that encodes the intended semantic mapping. For specifying semantic mappings on an abstract level, we provide a graphical user interface. In this interface, the way the data values are interpreted (i. e., the fuzzy membership function) is defined through analogy (Section 3.3.2). The mapping to a visual attribute is determined by rules which are graphically specified (Section 3.3.3).

A key component of our framework is the process of semantic shader augmentation, which is further illustrated in Fig. 3.4. The input shader in this example is a raycasting program that depicts a volume in a sparse rendering style. The code segment at the top stores the volume z-coordinate in the 'fCoordZ' variable. The segment at the bottom writes the level of sparseness to the 'fSparseness' variable. The level of sparseness is controlled by a single 'fParam' parameter. The sparseness parameter in-

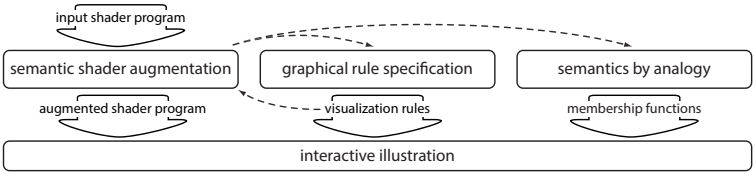


Figure 3.3: General schematic overview of our framework for semantics by analogy. Three components influence the resulting interactive illustration. The boxes in the center row show the components' main processes. The thick arrows leaving the processes represent the entities which influence the result. The dashed lines indicate the interrelations between the three components.

fluences the saturation of the color, the usage of a diffuse and specular shading term, and the use of contours. The two variables are marked as a semantic property and as a visual attribute using dedicated tags. The semantic shader augmentation replaces the static mapping of sparseness with a rule-based mapping. The application of the sparse rendering style is then controlled by two visualization rules, which depend on the volume z-coordinate. To achieve this, the process of semantic shader augmentation injects new functionality into the *input shader program*. This results in an *augmented shader program* which is capable of rendering visualizations of semantic properties, and of evaluating semantic visualizations rules. The rules used in the shader augmentation, as well as the membership functions, can be interactively explored and specified in a graphical user interface (Fig. 3.5).

The described semantic shader augmentation allows to quickly derive a case-specific visualization system from an initial shader program in a collaborative session of a visualization expert and a domain expert or an illustrator. The domain expert or illustrator is then provided with a graphical user interface for exploring and specifying semantic mappings on an abstract level. Together, these processes add more direct control and flexibility to rule-based rendering and also permit users to explore the mapping space flexibly and dynamically.

3.3.1 *Semantic Shader Augmentation*

Shader programs usually encode a certain visual abstraction. For example, a shader can perform illumination computations, can render contours, or perform illustrative volume rendering techniques. All of these techniques typically specify a static visualization mapping. In volume rendering, e. g., this mapping is modeled as a transfer function. By *automatically* augmenting the input shader, we replace the static visualization mapping with a dynamic rule-based mapping, while the general visual abstraction

such as 'illumination,' 'contour depiction,' or any illustrative volume-rendering technique is maintained.

We can work with arbitrary visual abstractions in our approach, but we continue to use the example of a GPU raycaster that generates images of different levels of sparseness. This program takes a volume texture as input, aggregates color samples along a ray, and outputs a resulting color value that depends on the pre-defined sparseness parameter. The color samples are computed by reading a scalar value from the volume, executing a transfer function look-up to map the scalar value to a color, and performing the sparseness computation. This means that input data-values (i. e., scalar values) are mapped to output visual-attributes (i. e., sparseness) in a static visualization mapping. For automatically adding to this initial shader rule-based rendering functionality that is more flexible, we use variables in the shader code as input and output parameters of a dynamic mapping. This results in the possibility to apply the sparse rendering method in a selective and flexible way.

In order to realize this augmentation we perform two *automated* steps: (1) we render images of the data properties so that users can define semantics by analogy and (2) we add functionality to evaluate fuzzy visualization rules. These steps relate to two distinct types of semantic entities. Step 1 refers to semantic data properties which form the input, i. e., the domain to our visualization mapping. For instance, these are properties such as the volume z-coordinate or the normal in the raycasting shader. The other type of semantic entities in Step 2 are visual attributes which represent the output, i. e., the co-domain of our mapping. Examples of these attributes are color, opacity, or parameters of sparseness or stylized shading. As depicted in Fig. 3.4, the *input shader program* is automatically extended, resulting in an *augmented shader program* which is capable of performing these two steps.

The declaration of properties that form the input and output of the rule-based mapping is realized using dedicated tags in the shader code. An example for such a tagged variable is shown below:

```
<SemanticProperty name="z_coordinate"
                  variable="fCoordZ"
                  type="float">
   float fCoordZ = vecRayPosition.z;
</SemanticProperty>
```

The tag contains information about the semantic entity: a specifier, the name, and the type of the associated variable. Similar tags can be added for the visual attributes that are encoded in a shader program. By adding such tags to a number of variables of interest, users can declare which variables in the input shader are used as either semantic data properties or as visual attributes. This consequently enables users to

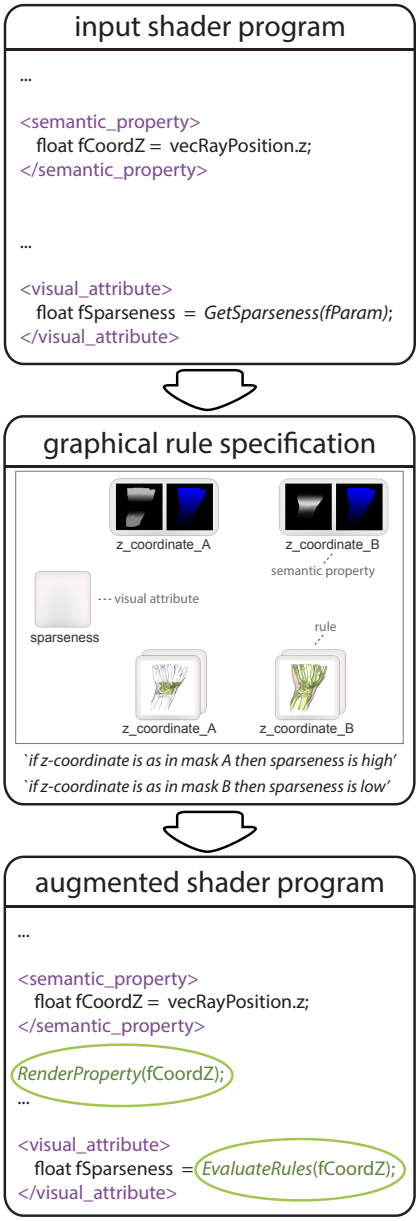


Figure 3.4: Overview of the concept of semantic shader augmentation. Arbitrary variables in an input shader program are defined as input and output to a semantic visualization mapping by adding dedicated tags to the shader code. The program is then automatically augmented with functionality which allows to specify contributions of input data-properties by analogy, as well as with functionality to realize a mapping of output visual-attributes that satisfies visualization rules.

dynamically explore the usage of different input and output parameters in the visualization rules specified later. In addition, by using a shader file that has been tagged previously, users can also work with pre-defined input and output features.

An application scenario of this shader augmentation is the rapid development of a case-specific visualization system in a collaborative session of a visualization expert and a domain expert or an illustrator. One case in this scenario is that the domain expert or illustrator has an a-priori understanding which data values or data derivatives are of interest for the desired visualization. For example, an illustrator might be interested in assigning a specific rendering style based on the surface orientation. In this case our method allows the visualization expert to quickly derive a suitable visualization system from the input shader, simply by adding tags to the respective variables in the shader code. Another case in this scenario is that the domain expert or illustrator has no a-priori understanding which data entities are of interest for the intended visualization. For example, a medical researcher might be interested in visualizing interior structures within a volume, but is not aware which variables in the shader code can be used for this task. In this case our method provides the possibility to collaboratively identify semantic variables that can be used to achieve the desired results.Identifying and tagging suitable semantic variables can still be a rather time-consuming process, but it has to be performed only once. As soon as suitable variables are tagged by the visualization expert, the domain expert is provided with an abstract control for using these variables in rule-based mappings. This abstract control of the shader program is given by the graphical user interface described in Sections 3.3.2 and 3.3.3. Using this interface does not require any programming skills, while it provides the domain expert with a flexible control over GPU-based semantics-driven visualizations.

To augment the input shader automatically we create a copy of the shader file in main memory, extend it with the required functionality, and use the resulting code as a dynamically loaded shader program. This also means that the input-shader code can be edited during runtime. The code generation starts with parsing the source file to detect the tags that define variables as semantic entities. The required functionality is then automatically inserted at the locations indicated by these tags. This dynamic shader augmentation is one of our key novelties to the original semantic-layers approach [Rautek et al., 2007, 2008a]. There, the rule-based rendering functionality is required to be hard-coded in a volume rendering program, and is thus present a-priori. This restricts the possibilities of quickly exploring different semantic mappings for different visualization techniques, because each of the techniques has to be made usable for the rule-based rendering individually.

The novelty of our semantic shader augmentation is not the dynamic code generation as such. It is the way we employ the well-known tool of dynamic shader generation to realize a dynamic specification of the mapping of arbitrary data values to arbitrary visual attributes based on illustration rules. Furthermore, our method for dynamic shader generation differs from the concepts of the *UberShader* [Hargreaves, 2005] and the *SuperShader* [McGuire, 2006]. Both of these concepts use pre-coded fragments of shader code that are combined to a new shader program on the fly. We, in contrast, do not use pre-coded program fragments in our dynamic code generation process. The code fragments which we inject into the shader are entirely generated on the fly. This means that our method can be used to augment any existing shader program with a minimal implementation overhead. The only code which has to be added to the input shader manually are the described tags, the remainder of the code is automatically generated. This approach drastically reduces the necessary implementation time for adding the required functionality to a given input shader, and also improves the code's readability.

In order to realize Step 1 of the augmentation, we automatically insert functionality into the shader that generates visualizations of semantic data properties and renders the result to a texture (e. g., Fig. 3.6 left). Here, we use one color channel of the output texture per dimension of the rendered data property: for one-dimensional variables we use the blue color channel, for three-dimensional variables we use all three channels. To generate the image we perform raycasting, composite the values that are assigned to the semantic variables during the execution of the input shader, and map the composited value to a visible color range. For the compositing, users can select one of three methods: averaging, maximum, or slicing. The semantic-property image in Fig. 3.6 to the left, for instance, displays the maximum density. To map the composited value to a visible data range, we multiply it with a user-adjustable factor. Although more advanced mappings could be used, we found the described simple ones to be sufficient for our purpose. Furthermore, it allows us to operate on only one color channel when one-dimensional properties are processed, saving computational resources. A disadvantage of this simple mapping scheme, however, is that it ignores negative values. The simple multiplication with a positive factor does not map negative composited values to positive color-values. This could be remedied, e. g., by shifting the range of the composited values. But this would result in mapping a composited value of zero to a gray color value. As we wanted to depict a composited value of zero with black color, we decided to not shift the range of the composited values. Another possibility for visualizing the composited values would be to use an additional 'transfer' function that maps the composited values to arbitrary color schemes. However, we did choose to employ the described mapping for simplicity.

The average and maximum compositing represent an extension to the semantic-layers approach. They map semantic properties from object-space to image-space. An application of these *image-space semantics* is demonstrated in Section 3.4.

In Step 2 of the augmentation, we automatically enhance the shader with the capability to evaluate a semantic rule base. For this purpose we implement the fuzzy-logic arithmetics as described by Rautek et al. [2007, 2008a]. They use two types of membership functions, related to data properties and to visual attributes. In the original approach, pre-defined functions are employed for both types. The membership functions of the visual attributes define the mapping from membership-function values of a data property to values of a parametrized visual attribute. In our approach, we work with a set of functions that are defined through a simple function editor for the visual attributes. For the semantic data properties, on the other hand, we use dynamic membership functions which are specified by analogy as described next.

3.3.2 *Semantics by Analogy*

Users of our system can define data semantics by brushing values of interest in the visualizations of data properties created by the semantic shader augmentation. This specification of data semantics through brushing is one of the two interactive graphical processes for achieving the intended visualization, as depicted in Fig. 3.5. Data semantics in this context refers to the way data values are mapped to visual attributes by means of membership functions of fuzzy sets. During brushing, these membership functions are adjusted according to the selected color values. This concept allows the user to see and directly assign a meaning to the data ranges of interest. The user can, therefore, visually draw conclusions about which data ranges are meaningful for the visualization he or she intends to achieve. In this way an appropriate mapping of these data ranges to visual attributes can be defined explicitly. For example, the user might be interested in areas of low density as in the example in Fig. 3.6. These areas of interest are visible and can be directly selected in the maximum density image. This is what we denote as the specification of *semantics by analogy: the resulting visualization is visually analogous to the marked data ranges*. The described way of specifying a semantics-driven visual-mapping happens dynamically because the membership functions are constantly updated during the brushing process. The specification by analogy gives a more direct control of the semantics than the use of membership functions specified with a function editor. It is, thus, the second key novelty that we propose as an extension to the semantic-layers technique by Rautek et al. [2007, 2008a].

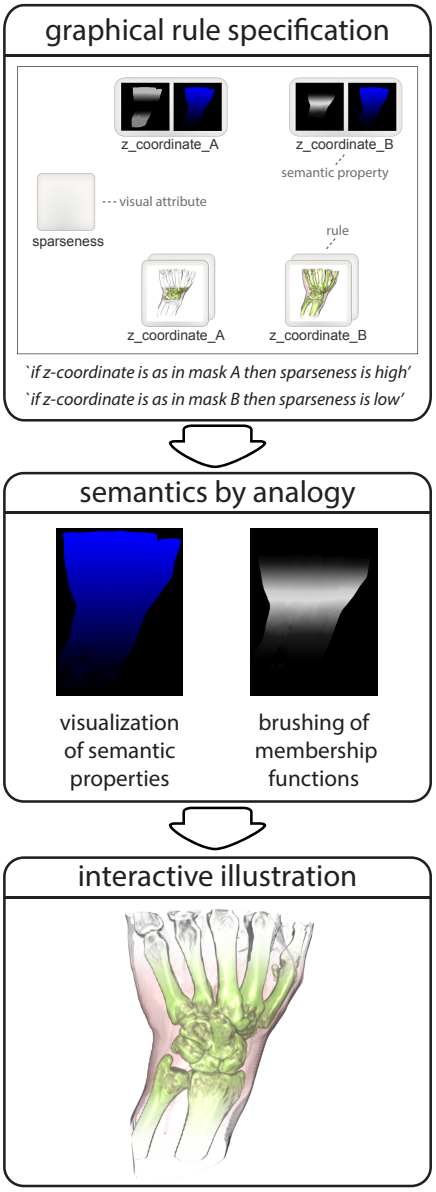


Figure 3.5: Conceptual overview of our technique for a graphical interaction with semantics-driven visualizations. Graphically specified visualization rules (left) make use of data semantics which are defined by analogy, i. e., by defining membership functions for semantic data properties via brushing on renderings of these properties (center). This technique allows a domain expert to create interactive illustrations (right) on an abstract level, while the underlying entities are automatically derived from an input shader program as illustrated in Fig. 3.4.

Further, in the original approach by Rautek et al. [2007, 2008a] only 1D properties are used due to the complexity involved in specifying 2D or 3D membership functions with a function editor. In contrast to this restriction to 1D properties, our approach of defining membership functions by brushing on a color image natively supports the definition of 2D and 3D membership functions. This facilitates the use of data derivatives such as the normal or the curvature direction in semantics-driven visualizations. Fig. 3.12 shows an example.

The concept of rendering data properties to intermediate buffers and using these buffers as the basis for the generation of stylized renderings is directly related to the seminal concept of G-buffers proposed by Saito and Takahashi [1990]. However, we use the intermediate buffers in a different manner than Saito and Takahashi [1990]. While Saito and Takahashi [1990] perform 2D image processing on the G-buffers to generate renderings, we use the buffers to interactively specify membership functions which then influence the resulting rule-based visualizations.

The membership functions are implemented as single-channel texture images. The dimensionality of the textures is determined by the number of dimensions of the represented data properties. We here discuss the 1D case as illustrated in Fig. 3.6: a one-dimensional data attribute that is associated to a one-dimensional membership-function texture. The definition of 2D and 3D membership functions is implemented analogously by simply adding additional color channels to the process. The domain of a membership function represents the scalar values of a semantic property, which in our case are given by the color values in the semantic-property image. The co-domain of the mapping are the corresponding membership-function values in the range of [0,1]. In our realization of the brushing mechanism we provide visual feedback about the current membership function by means of a gray value mask (Fig. 3.6, right). This membership-function image is generated by mapping the color values of the semantic-property image to the corresponding membership-function values (Step 5 in Fig. 3.6).

For brushing into the membership function, we treat the function as a histogram of the color values that are marked by the user. When the user brushes, we sample the color value at the cursor location in the source image (Step 1 in Fig. 3.6), map the sampled color value to the domain of the membership function (Step 2 in Fig. 3.6), and increment the membership-function value at this location (Step 3 in Fig. 3.6, modification through adding). We employ an adaptive brushing behavior that lets the membership function shift to the recently marked data range (Step 4 in Fig. 3.6). This adaptive selection permits users of our system to rapidly explore different data ranges without having to deselect formerly marked ranges to switch the focus to the recently selected data range. For realizing the adaptive brushing, we apply a normalization of

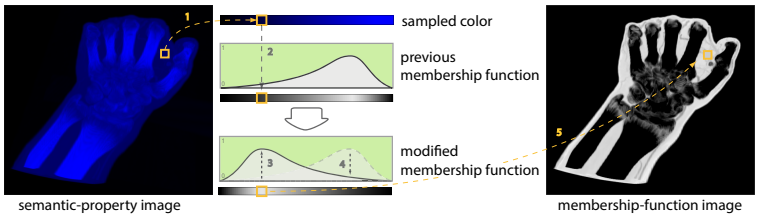


Figure 3.6: Brushing on a semantic-property image (left: maximum density image) to specify a membership function (center), and generating a membership-function image or mask (right). To brush into a membership function we sample the color at the cursor location (1) and map this color to the domain of the membership function (2). The previous membership function is incremented at the corresponding location (3), a normalization causes the modified membership function to decrease in other data ranges (4). To generate the membership-function image from the semantic-property image, we sample the membership-function value at the location corresponding to a pixel's color and write this value to the mask (5).

the membership function so that the area below the function equals to one. This normalization ensures that the current membership function represents a normalized probability distribution of brushed color values. This means that the membership function adapts to the frequency of the brushed color values. As an alternative, we also provide a method to subtract from the membership function, which is implemented analogously to adding. The subtractive brushing allows users to adjust the current selection by lowering the membership-function values in particular data ranges. Finally, we allow users to modify the membership function in a non-adaptive way. In this case, we skip the normalization step, so that brushing adds to or subtracts from the membership function at only one color range without affecting other ranges.

The described brushing mechanism is related to the concept of dual-domain interaction introduced by Kniss et al. [2002]. This concept defines interactions which link the spatial domain of the resulting volume rendering with the domain of the transfer function. The brushing interaction presented here, in contrast, takes place in an intermediate domain. The brushing in our approach is not performed on the final visualization as the probing used by Kniss et al. [2002]. Neither does it influence a conventional transfer function. However, the brushing in our approach is performed on visualizations of semantic properties, which do lie in the same spatial domain as the resulting visualization, but represent values from an intermediate abstract domain. Further, the control of the membership functions through brushing is related to the control of transfer functions. But again, the membership functions do represent an intermediate abstract domain. They only have an effect on the resulting visualization in combination with a set of illustration rules.

In order to allow users to broaden the data range of input (color) values that are mapped to non-zero output (membership function) values, we apply a smoothing operation to the membership function. This smoothing is realized by means of computing a mipmap of the membership-function texture so that users can freely control which mipmap-level to use for brushing. We observed, however, that the standard mipmap generation 'averages out' the membership-function values. This means that membership-function values would decrease with descending mipmap levels as they get averaged with zero-valued neighbors. To avoid this negative effect, we create an accumulative mipmap: we compute the value of a pixel in a smaller mipmap image by summing up the values of the corresponding pixels in the associated larger mipmap image, instead of using the average. This procedure allows us to maintain the magnitude of the membership-function values throughout all mipmap levels.

In addition, the interpolation of texture-border pixels with a border color when performing a texture lookup requires to employ texture coordinate clamping. This is always necessary when we access the membership function texture or one of its mipmap levels, e. g., when we use the color values from brushing as texture coordinates to look up values in the membership function. To cope with this effect we discard the border regions by clamping the texture coordinates to the central range of the membership function. When performing a lookup in the mipmap, we determine an upper and lower texture-coordinate limit from the current mipmap level and the size of the membership-function texture and clamp the texture coordinates accordingly.

3.3.3 *Graphical Rule Specification*

The semantic-property images enable users to dynamically specify a property's contribution to the visualization mapping by analogy. We also want to facilitate the direct and flexible construction of illustration rules that make use of these properties. We realize this control with a rule specification via a graphical user interface (Fig. 3.7, 3.8). As illustrated in Fig. 3.5, this rule-specification interface is one component for graphically interacting with visualization semantics. The other component is the brushing interface for defining semantics by analogy, as described in Section 3.3.2. Together, these two components can be employed to provide domain experts with a direct control of semantics-driven visualizations, abstracting from visualization and programming expertise.

The result of the graphical rule specification is a set of textual rules, such as *'if diffuse illumination is high then opacity is low.'* This semantic rule base is constantly evaluated by the augmented shader. The rule's result can, therefore, be dynamically generated. Changes to the member-

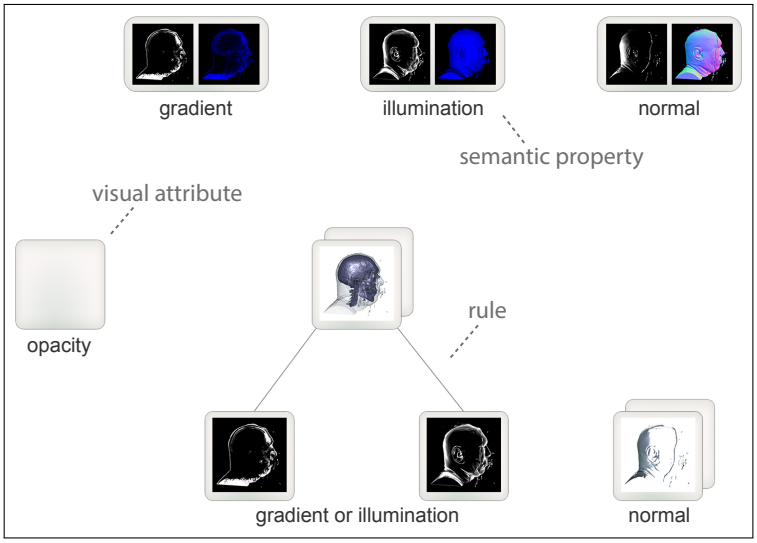


Figure 3.7: The interface for graphical rule specification. The widgets in the top row represent semantic data properties. They are used for switching between data properties, for creating rules, and for extending rules. Rules are depicted as expression trees in the bottom part. The nodes of the expression trees are interactive elements for combining and extending rules. The widget to the left represents a visual attribute. It serves for using this visual attribute in an illustration rule. Fig. 3.8 shows an interaction sequence.

ship functions resulting from brushing and changes in the rule setup can immediately be shown in the interface. This opens up new possibilities for the exploration of semantics-driven renderings.

The interface for rule specification shown in Fig. 3.7 contains three types of widgets which show preview images of the semantic entities they represent. These semantic entities are either a data property (top), a node of an expression tree (bottom), or a visual attribute (left). Fig. 3.8 shows an interaction sequence for creating a rule. The widgets located in the upper part of the interface represent the input data properties (Fig. 3.8a). Such a semantic-property widget contains both the semantic-property image as well as the membership-function image. By dragging a connection between two such widgets the user can graphically specify a rule based on the two respective properties (Fig. 3.8b). To allow users to select a logical operator that combines the two properties, we use a design gallery [Marks et al., 1997]. We render an array of result images for rules with different combinations of the three operators *AND*, *OR*, and *NOT*. The design gallery in the example in Fig. 3.9 shows result images for six different logical combinations of the expression *'z-coordinate is peripheral'* with the expression *'density is low'* (e. g., *'z-coordinate is peripheral AND density is low,' 'z-coordinate is peripheral OR density is low,'*

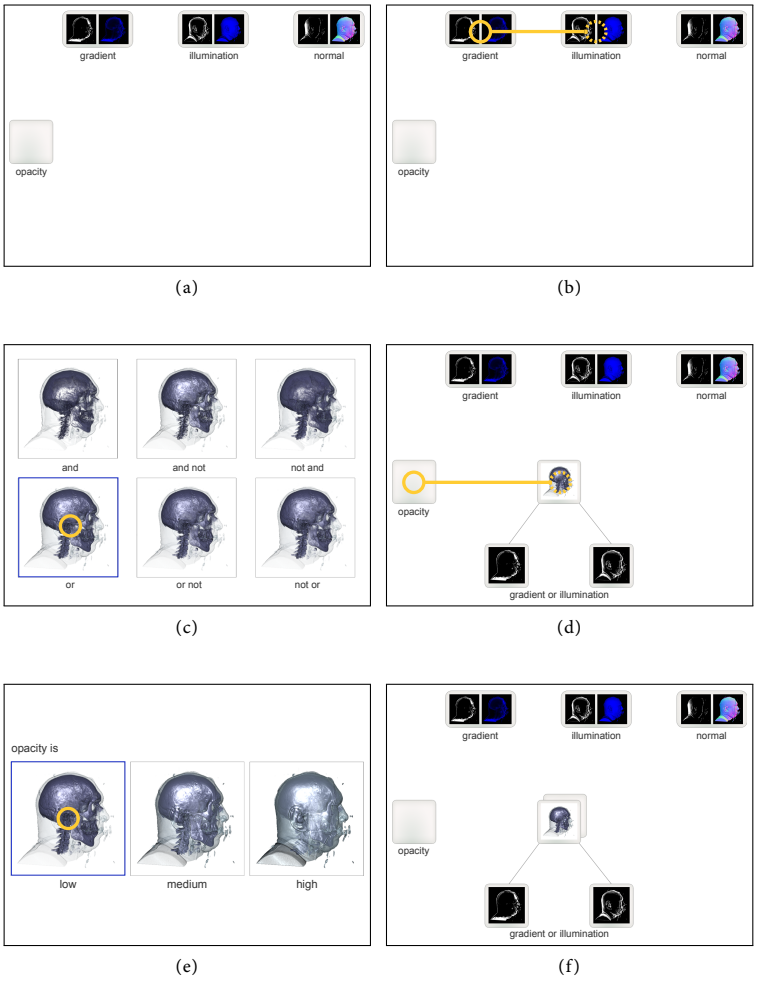


Figure 3.8: An interaction sequence for the graphical specification of an illustration rule. (a) The widgets located in the upper part of the interface represent the semantic properties. (b) A rule based on two properties is defined by dragging a connection between two such semantic-property widgets. (c) The interface switches to a design gallery. It depicts the results of combining the two chosen properties with different logical operators. A logical operator for the rule is selected by clicking on a preview image. (d) The newly created rule appears as an expression tree at the bottom part of the interface. Then, a visual attribute is assigned to the rule by dragging a connection from the visual-attribute widget on the left hand side to the root node of the expression tree. (e) A design gallery appears which allows the user to select a visual attribute membership function. (f) After the selection has been performed, the interface shows the complete visual representation of the specified rule.

'z-coordinate is peripheral OR density is NOT low,' ...). The visual attribute used in the combined rule is *'sparseness is high.'* Using the *AND* operator in this example results in a sparse rendering of soft tissue in peripheral z-coordinate regions. Using the *OR* operator depicts all soft tissue as well as all peripheral z-coordinate regions with the sparse rendering method. By clicking on the desired result image, users can specify which operator they want to include in the rule (Fig. 3.8c). Once a rule is specified in the described way, a visual representation of the rule is created as an expression tree, which is located at the bottom part of the rule-definition interface (Fig. 3.8d).

The nodes of this graphical expression tree are the second type of widgets we employ. Every leaf-node widget of the tree represents one semantic property, and displays the corresponding membership-function image. Every interior-node widget represents a rule or sub-rule and displays the result image generated by evaluating this rule. The expression-tree widgets are interactive elements for extending and combining rules. Drawing a connection from a semantic-property widget to an expression-tree widget allows users to extend the rule represented by the expression tree. The semantic property is included as an additional operand in the rule. By drawing a connection between two root nodes of different expression trees, the user combines the two affected rules to form a single rule. The described rule manipulations are implemented as string operations on the rule, while the graphical interface elements are generated from the modified textual rules.

The third type of widgets we use in our interface for rule specification represents the visual attributes that can be included in rules. These widgets are located at the left part of the interface. They are used to assign a visual attribute to a given illustration rule by dragging a connection to the root node of an expression tree (Fig. 3.8d). The assignment of a visual attribute to a rule results in exchanging the *then* part of the rule to include the selected visual attribute.

A visual attribute can be associated with a set of pre-defined visual attribute membership functions. Each function describes a different mapping from an aggregated membership-function value to a value of the parametrized visual style. To select one of these visual attribute membership functions for a rule, we also use a design gallery (Fig. 3.10). We present the user with a set of result images generated by the current rule using different visual attribute membership functions. The user chooses the desired function by clicking on the corresponding preview image (Fig. 3.8e). Once a visual attribute is assigned to a rule in this way, we render an instance of the visual attribute widget and attach it to the root node of the rule's expression tree (Fig. 3.8f). The graphical rule specification interface is our third major extension to the original semantic-layers approach by Rautek et al. [2007, 2008a].

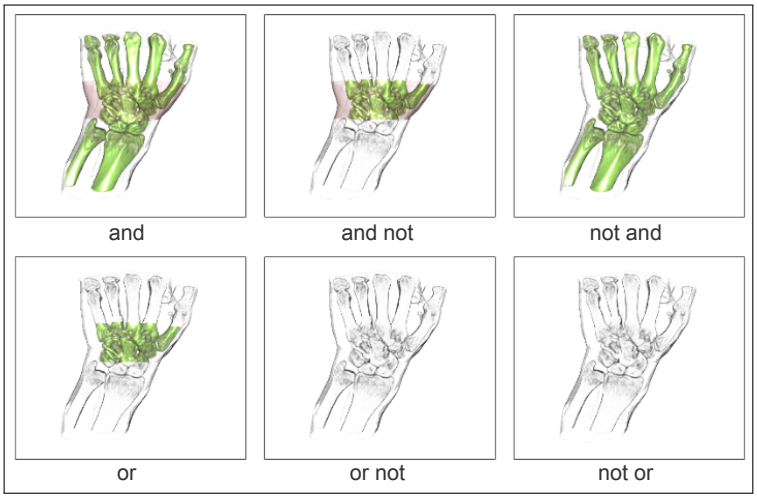


Figure 3.9: Design gallery with images of possible logical operations for adding a second rule to the rule from the example in Fig. 3.11f. The rule used for generating Fig. 3.11f is *'if z-coordinate is peripheral then sparseness is high.'* Herein, *'peripheral'* regions of the property *'z-coordinate'* are defined by a corresponding mask. The second rule that we add in this example uses the density as a semantic property. We create a mask on this semantic property that covers the regions of low density, similar to the mask created in the example in 3.6. This procedure adds a second visualization rule, which is *'if density is low then sparseness is high.'* When adding this second rule, the user is presented with a design gallery that shows the results of different combinations of logical operators for combining the existing rule with the newly added rule. The combined rules that correspond to the previews above are (in reading order): *'if z-coordinate is peripheral* AND *density is low then sparseness is high,' 'if z-coordinate is peripheral* AND *density is* NOT *low then sparseness is high, 'if z-coordinate is* NOT *peripheral* AND *density is low then sparseness is high,' 'if z-coordinate is peripheral* OR *density is low then sparseness is high,' 'if z-coordinate is peripheral* OR *density is* NOT *low then sparseness is high,' 'if z-coordinate is* NOT *peripheral* OR *density is low then sparseness is high.'*

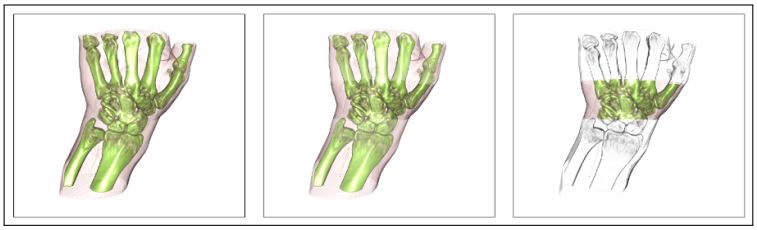


Figure 3.10: Design gallery with preview images of possible visual attribute membership functions for the example shown in Fig. 3.11f.

3.4 RESULTS AND DISCUSSION

In this section we show explanatory examples of the variety of visualizations that can be achieved with our approach. Fig. 3.11 demonstrates a simple case of semantic shader enhancement and the selective application of a rendering technique by analogy. In this example, we intend to highlight the central region of the dataset. As an input shader we use a volume renderer that produces images of different levels of sparseness. Fig. 3.11a shows a rendering for a high sparseness parameter, while Fig. 3.11b depicts a visualization for a low sparseness parameter. This shader is used as input to our system to demonstrate the easy enhancement of existing shaders. In the shader file we tag the data attribute *'z-coordinate'* as semantic property. This is the z-component in 3D texture coordinates as they are typically used in raycasting-based volume renderers. Furthermore, we tag the visual attribute *'sparseness'* that is specifically used in this shader. Interactively adding the two simple rules *'if z-coordinate is as in mask A then sparseness is high'* and *'if z-coordinate is as in mask B then sparseness is low'* results in an *automatic* augmentation of the shader. Now we define the regions that we intend to highlight by analogy, i. e., by brushing on the z-coordinate image shown in Fig. 3.11c. For rendering peripheral z-coordinate regions with high sparseness, we brush the semantics according to the first rule to mask A in Fig. 3.11d. This results in the image shown in Fig. 3.11f. In order to render the central area with low sparseness, we add the data semantics for mask B in Fig. 3.11e by brushing on the central area of the z-coordinate. This results in the image shown in Fig. 3.11g. This example demonstrates that the enhancement of the shader is simple yet flexible. For example, the smooth interpolation between regions of high sparseness and low sparseness in Fig. 3.11g is achieved with only two rules. In addition, defining semantics by analogy allows to directly apply the sparse rendering to intended regions.

In the example of Fig. 3.12 we show a simple case of deriving a focus-and-context renderer from an existing raycasting shader. We demonstrate how our approach supports the goal to selectively depict different structures in a dataset (here: bone and soft tissue) with different visual attributes. The selective application of these visual attributes shall be controllable based on semantic properties derived from the data (here: directional information). In order to achieve this goal, we take a simple shader program as input that renders the image with two different transfer functions (one for contextual rendering and one for focus rendering). We then use the two rules *'if average xz-gradient is as in mask A then rendering is contextual'* and *'if average xz-gradient is as in mask B then rendering is focused'*. Note that the keyword *'average'* is a further extension to the semantic-layers approach. It refers to a generalization of semantic properties from object-space to also include image-space prop-

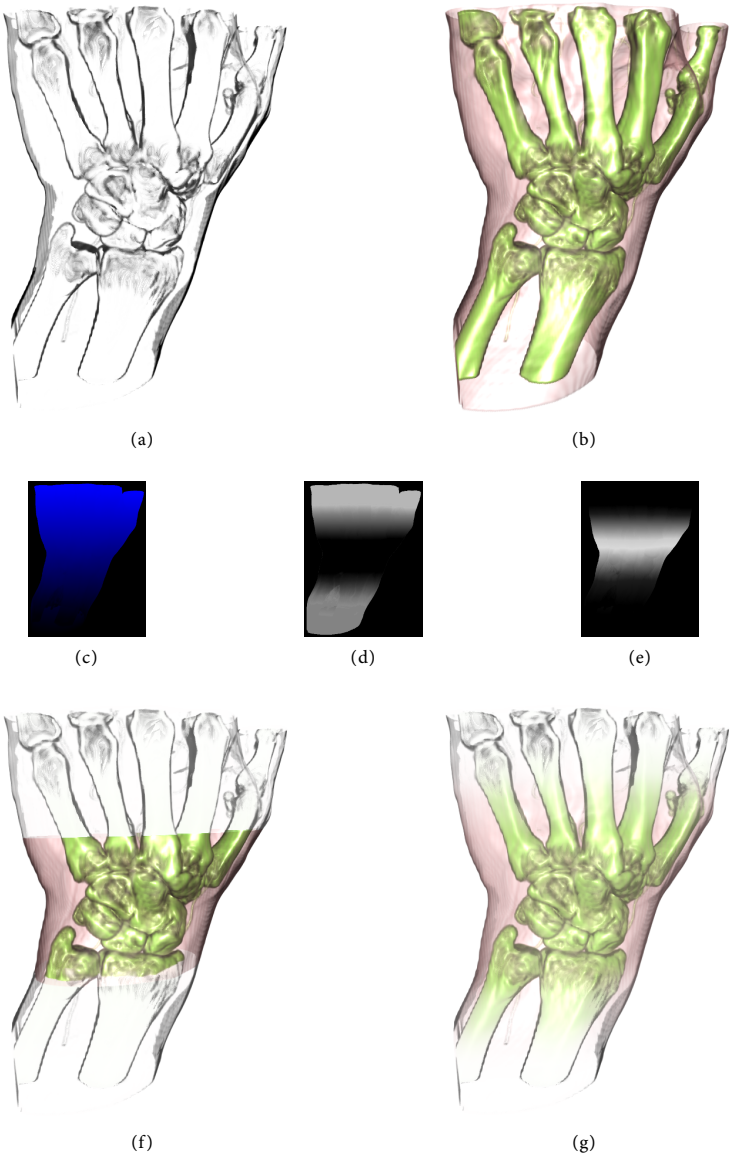
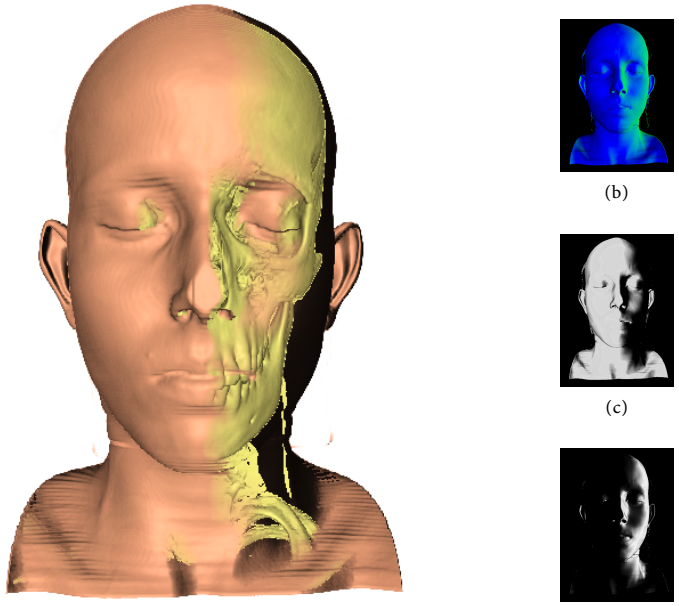


Figure 3.11: Augmenting an input shader that produces images of different levels of sparseness. The top row shows renderings (a) for a high and (b) for a low sparseness parameter. In the augmented shader, we control the sparseness parameter with two masks that are defined on the volume z-coordinate. Brushing on the (c) z-coordinate image yields (d) mask A and (e) mask B. We achieve a smooth interpolation of sparseness with the rules (f) *'if z-coordinate is as in mask A then sparseness is high'* and (g) *'if z-coordinate is as in mask B then sparseness is low'* as an additional rule.



(a) (b) (c) (d)

Figure 3.12: Controlling contextual and focus rendering modes via the property 'average xz-gradient.' (a) The result is generated from the two rules *'if average xz-gradient is as in mask A then rendering is contextual'* and *'if average xz-gradient is as in mask B then rendering is focused.'* The right column depicts (b) the *'average xz-gradient'* image as well as (c) mask A and (d) mask B.

erties. In object-space, the fuzzy logic evaluation is done on a per-sample basis while in image-space it is done on a per-ray basis. To demonstrate this generalization, we here use the image-space property *'average xz-gradient'*. Fig. 3.12b depicts the rendering of this semantic property. The *'average xz-gradient'* is the average of the gradient taken along the ray which is transformed to image-space and which is often used, for example, for shading calculations. We use the x- and z-coordinates of this property to interpolate between the two rendering modes, i. e., context and focus visualization. This allows us to selectively apply the two different rendering modes based on the surface orientation. The two masks created to control the interpolation between the two modes are shown in Fig. 3.12c (mask A) and Fig. 3.12d (mask B). We use our method to apply the focus rendering on surfaces that are oriented towards the right as seen from the viewer.

The example in Fig. 3.12 shows a natural extension of the data semantics to a two-dimensional data property (*'xz-gradient'*). In the original semantic-layers technique it is necessary to use one-dimensional data semantics. This is due to the complexity involved in the specification

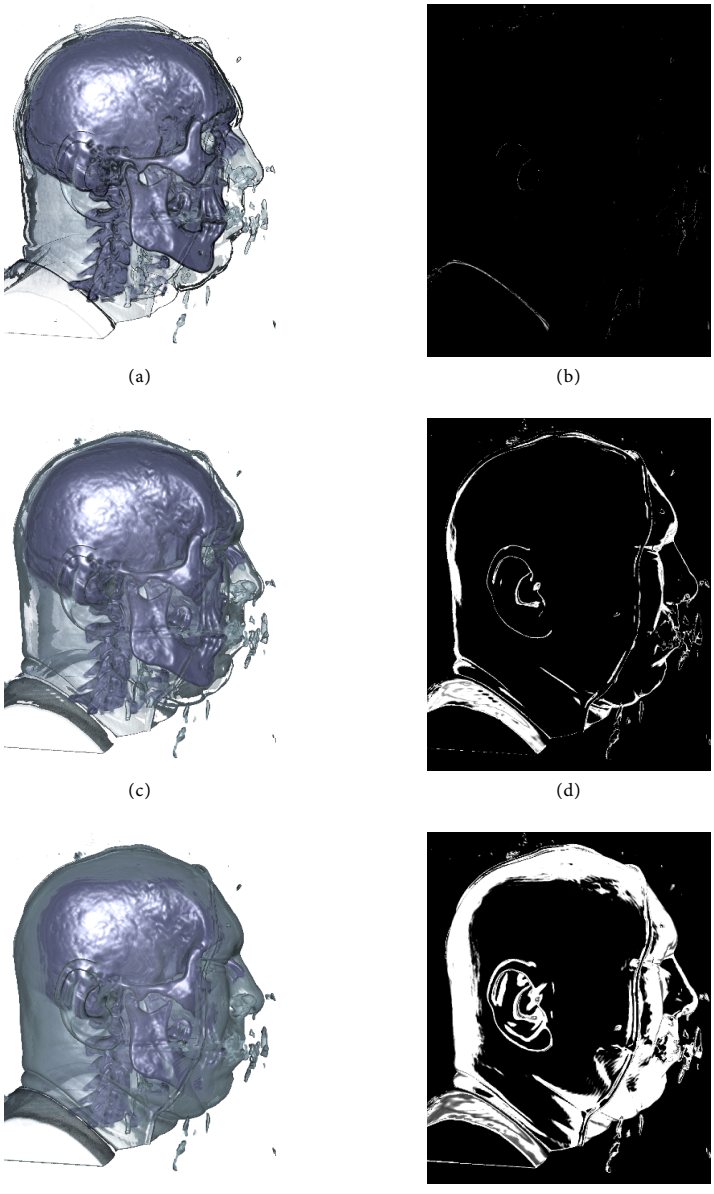


Figure 3.13: Using the semantics-by-analogy interface for an opacity modulation based on illumination intensity. We use the two rules *'if diffuse illumination is low then opacity is high'* and *'if diffuse illumination is high then opacity is low.'* The left column (a, c, e) shows different results that are generated with the masks in the right column (b, d, f) for the data semantics *'diffuse illumination is low.'*

of multi-dimensional membership functions. The functions are specified with a 1D function editor. With our approach, we can easily use the brushing on images of 2D and 3D properties to interactively and directly specify 2D and 3D membership functions, as exemplified in Fig. 3.12.

In Fig. 3.13 we demonstrate the progressive adjustment of a visualization using the semantics-by-analogy user interface. In this example we use the *'diffuse illumination'* term of a regular raycasting shader as a semantic property. Our goal in the augmentation is to map the diffuse illumination term to the opacity of each sample. This mapping allows us to make regions that are highly illuminated more transparent. The two rules *'if diffuse illumination is low then opacity is high'* and *'if diffuse illumination is high then opacity is low'* are used to achieve this effect. This visualization mapping is similar to the context-preserving volume rendering technique by Bruckner et al. [2006]. This demonstrates the ability of our approach to dynamically specify other (usually hard-coded) visualization techniques by the means of visualization rules. Fig. 3.13 shows the results (Fig. 3.13(a, c, e)) and the different masks that were brushed to generate these images (Fig. 3.13(b, d, f)). The brushing on the data semantics *'diffuse illumination is low'* as done in this example allows users to interact with the illustrative visualization and to adjust the illustration according to their needs and preferences.

The example in Fig. 3.14 further demonstrates the use of a semantics-driven opacity modulation. The goal of this visualization is to generate views on a skull-surrounded brain in an MR dataset of a human head. Raycasting of such a dataset is subject to occlusion of the brain by surrounding tissue. In order to cope with this occlusion, we apply our method to a standard raycaster to create see-through views on the brain. We achieve this by using the *'distance along the ray'* as a semantic property and the rule *'if distance along the ray is high then opacity is high.'* This setup allows a flexible application of opacity depending on the penetration depth. Similar to a clipping plane, it removes both the occluding skull and the brain tissue up to a specified penetration depth. This does not preserve the occluding brain tissue as a skull-removal scheme would do. In contrast to the static geometry of a clipping plane, however, this setup implements a dynamic semantics-driven clipping volume that can be interactively defined. This facilitates a flexible interactive specification of the visibility of different structures within the volume dataset. The left column in Fig. 3.14 depicts the results of applying this rule from different viewpoints (Fig. 3.14(a, d, g)). The right column shows the corresponding semantic-property images (Fig. 3.14(b, e, h)) and masks (Fig. 3.14(c, f, i)) that are used to generate the visualizations.

Fig. 3.15 shows more results of applying our technique to the visualization of the MR head dataset. We here again use our technique for a semantics-driven opacity modulation but use different input properties.

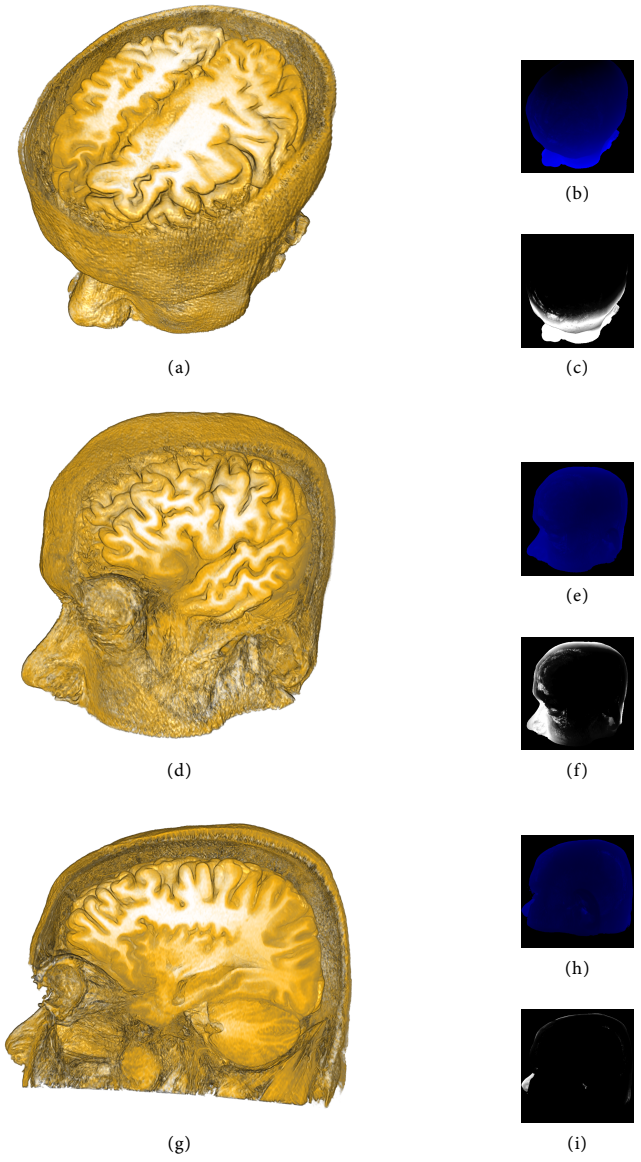
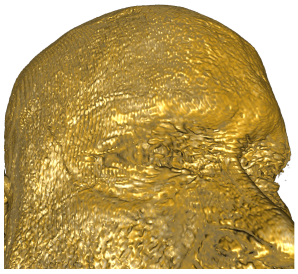
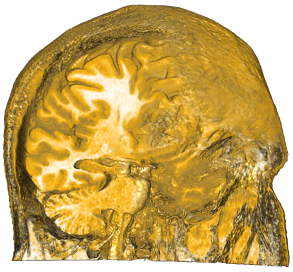


Figure 3.14: Generating views on a skull-surrounded brain in an MR head dataset. A semantics-driven opacity modulation generates see-through views on the brain. We use the rule *'if distance along the ray is high then opacity is high.'* This allows to apply opacity depending on the ray penetration depth, resulting in an interactive semantics-driven clipping volume. Similar to a clipping plane, this clipping volume removes all brain and skull tissue up to a certain ray position. It does not preserve the occluding brain tissue. The left column depicts results from different viewpoints (a, d, g). The right column shows the corresponding semantic-property images (b, e, h) and masks (c, f, i).

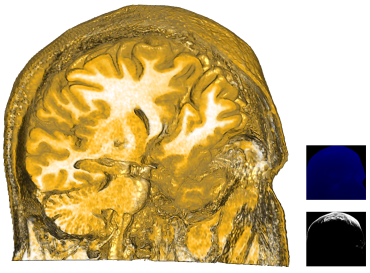
In the example in Fig. 3.15 we use the spatial object-space properties '*distance from object-space origin*' and '*object-space position*' to modulate the opacity in order to generate views on the brain. The initial raycasting shader without semantic shader augmentation (Fig. 3.15a) is subject to occlusion of the brain by surrounding tissue. We use the graphical control over the result of our volume illustration system to visualize the brain and to selectively visualize parts of the dataset.



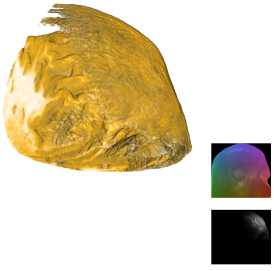
(a) Result of the initial shader. Without semantic augmentation, the shader is subject to occlusion of the brain.



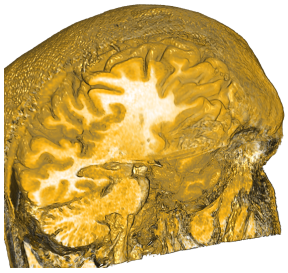
(b) Result of the augmented shader. The view on the brain is generated by using the two rules from Fig. 3.15c and Fig. 3.15d.



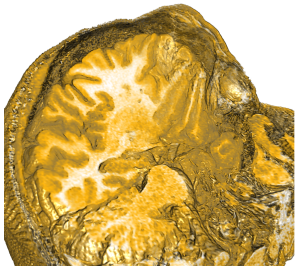
(c) Cut-away view created by using the rule '*if distance from object-space origin is high then opacity is high*.' The insets to the right show the property image and the mask.



(d) Selective view generated by using the rule '*if position is as in mask then opacity is medium*.' This is an example of a 3D semantic property.



(e) The result of Fig. 3.15b from a different viewpoint.



(f) The result of Fig. 3.15b from a different viewpoint.

Figure 3.15: Visualizations of the MR head dataset using an opacity modulation based on spatial object-space properties.

All results were generated in interactive sessions. The performance of our system depends on the set of rules, the used input shader, the current membership functions, and parameters such as screen resolution and sampling distance. For all results shown in this chapter we used a sampling distance of 0.5 voxels and achieved interactive frame rates on a dual core 3.2 GHz PC with a GeForce GTX 480 and 12 GB RAM. We used a 600×600 pixel viewport for the result image and for the rule-specification interface (which also includes the visualizations of the semantic properties and previews for sub-rules). The images shown in Fig. 3.11 were rendered at approximately 20 fps. The dataset in this example has a resolution of $122 \times 62 \times 128$ voxels. The example in Fig. 3.12 was rendered at approximately 15 fps, with a dataset resolution of $256 \times 256 \times 166$ voxels. For the results depicted in Fig. 3.13, we achieved a framerate of approximately 17 fps, rendering a volume of $128 \times 256 \times 256$ voxels. The framerate for rendering the images in Fig. 3.14 and Fig. 3.15 was approximately 8 fps, for a volume of $512 \times 512 \times 320$ voxels.

3.5 EVALUATION

We evaluated the proposed concepts by gathering user feedback. We conducted two separate user evaluations with two different target audiences. The first evaluation targeted medical domain experts, while the second evaluation examined the usefulness of our system for medical illustrators. Both evaluations were qualitative assessments of the benefits and drawbacks of our method. We designed the evaluations as participatory observational studies combined with contextual interviews. We decided to use a qualitative evaluation methodology because we consider a quantitative evaluation as inappropriate for validating our system. This decision is based on the discussion about the role of evaluation in visualization and human-computer-interaction (HCI) research. In this discussion, Carpendale [2008] advocates a more thoughtful application of a greater variety of research methodologies for evaluating information visualizations. She provides a survey of different evaluation methodologies and observes that quantitative methods can be prone to fault and to questionable validity for scenarios such as ours. Amongst other aspects, this is due to the fact that a quantitative experiment requires the rigorous control of many different factors, which are in fact difficult to control as a whole in situations like ours where complex and temporally long interactions are the essential elements of a system. Carpendale argues that a qualitative inquiry is more appropriate in such cases because it allows the researchers to consider the interplay among factors that influence visualizations, their development, and their use, and also to ground the studies in a more realistic setting. Along the same lines, Greenberg and

Buxton [2008] criticize the dogma of quantitative usability evaluation in HCI research. They stress the importance of choosing the appropriate evaluation methodology for a given research problem. They explain how the insistence on quantitative evaluation as a research methodology has fostered practices of weak science in HCI. Furthermore, Greenberg and Buxton [2008] also emphasize that premature usability evaluation can eliminate promising ideas in an early design stage. They recommend to learn from the ways design worthiness is validated in other disciplines, i. e., from the design critique as used in industrial design. We follow their recommendation by selecting a qualitative evaluation of our system involving design critique principles. One may argue that multi-dimensional, in-depth, and long-term case studies [Shneiderman and Plaisant, 2006] are best for evaluating creativity support tools such as ours. Although such an approach would be an appropriate method to validate our system quantitatively, the time required for such an evaluation approach makes it unfeasible for us to realize at this point.

3.5.1 *Feedback from Medical Experts*

The participants in the first evaluation were a neuro-ophthalmologist in a first session and a group of eight radiologists in a second session. The participants were chosen from contacts available to our research group. All participants volunteered to participate in the evaluation. All the participants were male and between 23 and 48 years old. Each session comprised an initial demonstration and explanation of the system, a guided experimentation by the participants, and a concluding semi-structured interview. The feedback of the medical experts was positive in general. They quickly understood the interface and were able to use it after a short instruction. For the neuro-ophthalmologist we prepared an example with opacity modulation. This example was very similar to the one shown in Fig. 3.14, but we made use of different semantic properties. As semantic properties we used the scalar values, the distances from the object-space origin, and the volume texture coordinates. The neuro-ophthalmologist particularly liked the flexibility of our system. He used the opacity modulation to generate cut-away views, and found this procedure more flexible than the usage of cutting planes. He stated that he could imagine to employ a system like ours for intervention-procedure planning and for teaching. The radiologists in the second session particularly liked the abstraction from visualization internals offered by our system. They also commented positively on the ease of use of the graphical rule specification. They suggested various extensions, for example to include a graphical library of pre-defined semantic properties and visual attributes from which the user can choose. The discussion with

the domain experts also revealed the benefit of the collaborative setting we described in Section 3.3. Semantic properties such as the *'xz-gradient'* or the *'diffuse illumination'* are non-trivial to formulate for most domain experts. But once such properties were visualized in our system and explained to the domain experts, they could easily understand and use them. On the other hand, it might be difficult for a visualization expert to figure out which data properties are useful for the domain experts. In our evaluation, it became obvious that a collaborative effort can help to find such useful properties.

3.5.2 *Feedback from Medical Illustrators*

The participants in the second evaluation were a graduate art student and two professional medical illustrators. The evaluation was conducted in three separate sessions. The participants were contacted after an internet research for medical illustrators in the Netherlands. All participants volunteered to participate in the evaluation. All the participants were female and between 26 and 42 years old. The art student is in a Master's program for interactive media and environments, and holds a Master's degree in graphics design as well as in fine arts. For simplification, we include her in the group of medical illustrators. The professional medical illustrators are both trained in fine arts as well as in medical illustration. One of them has several years of professional experience as a medical illustrator. The other has several years of professional experience as a photo retoucher and has been transitioning to medical illustration in the past seven years. None of the participants had noteworthy experience with volume data, but all had worked with renderings of polygonal 3D models. Thus, we started the sessions of this evaluation with a brief introduction to volume rendering. Apart from that, we applied the same methodology as in the evaluation with medical domain experts. We found out that some of the proposed concepts might indeed be of use for the target audience of medical illustrators. The participants in this evaluation could understand and use the interface quickly. Again, the feedback was positive in general. At the same time, the evaluation revealed some limitations of our system, which provide inspiration for future work. Interestingly, the feedback was highly congruent between the three participants.

The art student particularly liked the fact that when using our system, she could transfer knowledge and skills from graphics editing software. An example are our membership-function images which resemble the usage of masks in Adobe Photoshop®. She stated that the visual feedback provided by the semantic-property images makes it easier to decide on the application of rendering styles. She also liked the feature of working in the intermediate domain of the semantic-property images, while the

entire result is updated simultaneously. She stated that she clearly favors our graphical rule definition over a textual one.

The first medical illustrator gave clear confirmation that for her needs and preferences, the graphical rule specification is by far superior to a textual rule formulation. She liked our graphical approach to parameter specification, and stated that it would better meet the demands of 'visually oriented' persons than other more programming-oriented interfaces. She stated that she would use our proposed concepts in her work as a medical illustrator, given that they were integrated in a comprehensive system for illustrative volume rendering. Apart from that, she was very interested in direct volume rendering in general. She was particularly fascinated by the accuracy it provides. She emphasized that accuracy is crucial for medical illustration. She commonly uses a 3D atlas of polygonal models as a master to create illustrations. She stated that these atlases—in contrast to direct volume visualization—would lack a certain degree of precision because they involve the human factor of the artist creating the 3D models.

The second medical illustrator also revealed this keen interest in volume rendering as a reference for manually creating illustrations. Regarding our system, she also shared many of the views of the other medical illustrator. She confirmed that she clearly favors a graphical rule definition over a textual rule formulation. The illustration professional liked our use of masks, which allowed her to transfer skills from Adobe Photoshop®. She also stated that a system like ours could be of use for her work as a medical illustrator. She also commented very positively on the notion of using brushing as an interaction metaphor to modify the rendering, although she was not convinced by the benefit of brushing on the semantic-property images instead of the result image.

All medical illustrators gave rise to the question if it would be preferable to brush on the result image instead of the semantic-property images. They stated that they are used to work on either the image itself, or on a representation which is visually completely unrelated to the image, such as dialog windows. The art student liked the idea of working in an intermediate domain, but the two medical illustrators considered this more critically. They asked for the motivation of this design choice and said that they would be more familiar with brushing directly on the result image. They were not convinced that the visual feedback provided by the semantic-property images is crucial for obtaining the desired results. One of the medical illustrators even felt confused by the additional viewport needed for displaying and brushing on the semantic-property image. For her, it led to a confusion about which interaction has to be performed on which viewport. But both illustrators were convinced that with some training, they would get familiar with brushing in the intermediate domain. However, it would need a long-term evaluation to gain more insight about this learning process.

It is interesting that, in contrast to the medical illustrators, the domain experts did not raise the issue of brushing directly on the result image. They perfectly accepted the brushing on visualizations of data properties. In our opinion, this reflects that the group of medical domain experts is used to approaching the creation of an image in a data-oriented way. They are familiar with thinking of an image as the mapping of data. The illustrators, on the other hand, appeared to approach the creation of an image in a less data-oriented way—maybe because illustration based on real data has not been possible to a large degree until recently. Illustrators seemed to think more in terms of modifying the image directly, instead of modifying data or a mapping which underly the image. However, we assume that, with training, the illustrators would be able to transfer their creative skills to the world of data-oriented image creation.

3.6 LIMITATIONS

The one limitation identified in both of the two user evaluations was that the brushing mechanism can occasionally generate unexpected results. The brushing is influenced by a smoothing of the membership function, which we apply for broadening the range of marked data values. When the corresponding smoothing parameter is set to a high value, one can easily mark large data ranges. This turned out to have the negative side-effect of unintentionally selecting a too wide range when users intended to select a narrow data range. On the other hand, the creation of smooth volumetric masks is not possible without sufficient smoothing of the membership functions. Without smoothing, only one specific data value can be selected at a time, which leads to the creation of noisy or speckled masks. Our brushing mechanism, thus, involves a tradeoff between precision and execution time. With execution time, we refer to the time needed for the interaction of defining a desired mask, not the involved computation time. This tradeoff implies the drawback that our brushing interaction is prone to working in an either too precise and slow or in a too imprecise and fast way.

Another limitation of our method is the robustness of creating the semantic-property images, which form the basis for brushing. The final result is highly dependent on the semantic-property images. These images, in turn, are challenging to create for the general case. Especially for volumetric properties that are not related to iso-surfaces, it is hard to generate renderings which are well understandable and usable as a basis for brushing. In our examples we use either 2D or 3D properties that are directly related to iso-surfaces. In addition to this issue, the image-space selection of 2D and 3D data values can be problematic. Here, the maximum compositing cannot be applied, and component-wise averaging

results in introducing vectors which are not present in the dataset. With proper smoothing of the membership function, however, averages of 2D and 3D variables can still be used. We exemplified this in Fig. 3.12 with using the *'xz-gradient.'*

Working with image-space properties such as in Fig. 3.12 can also result in unexpected behavior. The domain experts here assumed to work with an object-space property and expected the mask to 'stick' to the dataset instead of following image-space directions. However, this unexpected behavior is due to the fact that the participants worked with the system only for a short time. We assume that such problems would be resolved quickly once that users become more acquainted with the system.

Another drawback of our technique is also related to the image-space selection of volumetric data values. The projection of scalar values to image-space requires compositing. When used in combination with a per-sample evaluation of the visualization rules in object-space, the composited scalar value can unintentionally differ from the per-sample scalar value. This then results in a discrepancy between the membership-function image and the final visualization. This discrepancy can be seen in the examples in Fig. 3.14, where the visualizations do not entirely match the membership-function images. The problem can only be avoided with a proper selection of semantic properties and rules, as well as with appropriate parameter tuning. This restricts the generality of our technique.

Furthermore, a general problem of our interactive graphical approach is the reproducibility of the results. Because interaction is required to specify the masks, it is hard to exactly reproduce results created earlier. This circumstance might let users perceive our system as being unreliable. However, the addition of the possibility to save membership functions and a comprehensive undo functionality may reduce the problem.

Finally, our technique does not address the problem of disjunct expert domains, although it addresses the challenge of semantics-driven parameter specification. A limitation of our approach is that the semantics we make use of do not originate from the problem domain (e. g., medicine), but rather from the solution domain (computer science). This implies that a domain expert using our system is likely not familiar with the semantics with which we provide him or her. The only aid we give for understanding the semantics are the semantic-property images. For example, a medical expert might have difficulties using the property *'distance along the ray.'* He or she might benefit much more from being able to use application-specific semantics such as *'lesion'* or *'tumor'* to steer semantics-driven visualizations.

3.7 CONCLUSIONS AND FUTURE WORK

In summary, we propose methods that improve the semantic-layers approach by Rautek et al. [2007, 2008a] in several essential ways. First, we introduce a semantic shader augmentation that increases the flexibility of the semantic-layers approach. It makes it possible to automatically *augment semantic shaders* at run-time. This concept can be employed, e. g., to enable a visualization expert to quickly derive a case-specific visualization system from an initial shader program according to the requirements of a domain expert. Second, we introduce the *semantics-by-analogy* approach. It allows users to brush properties to ease the process of defining and exploring data semantics. Third, we describe a user interface for the quick specification and exploration of fuzzy-logic rules. These two interactive graphical tools provide domain experts with a direct control of semantics-driven visualizations which abstracts from programming internals. Finally, we extend the semantic-layers method by introducing *image-space semantics*. These are incorporated in the semantic shader augmentation by using the keywords 'maximum' or 'average' in the specification of visualization rules.

One direction for future research is to address the above described problem of disjunct expert domains. It would be interesting to examine how our approach can be extended in order to permit the use of application-specific semantics.

Furthermore, the concepts presented here can be modified to allow users to brush directly on the result image instead of the semantic-property images. This interaction was also suggested by the medical illustrators in our second user evaluation. Many of the described limitations are arguments in favor of this idea. Our experiences with the hatching approach that is presented in Chapter 5 also suggest that the described direct brushing behavior is beneficial.

Apart from this, our approach makes progress towards a semantic markup of the whole volume rendering pipeline. The explicit specification of semantics in the volume visualization pipeline permits us to expose the system's underlying semantics to the user. This allows the user to directly interact with the semantics of the volume visualization process and, hence, to obtain a more direct control and a better understanding.

We see a large variety of application possibilities of our approach. Our approach can be applied to the rapid prototyping of volume visualization techniques. The proposed concepts can also be applied to the implementation of visualization frameworks that enable both visualization experts and domain experts to efficiently explore volume data and to generate powerful illustrative volume visualizations. We believe that the flexibility of our approach and the novel user interface make our approach a basis for a large number of possibilities in volume visualization.

Although our approach is very flexible, the initial manual tagging of shader files is still time-consuming. However, this process only has to be done once and each tagged shader can be re-used for further datasets or visualization problems without further tagging. Moreover, we plan to explore more flexible approaches in the future that allow the browsing of a shader file while automatically getting suggestions for data semantics. We believe that the user experience can be greatly improved with such a browsing extension. Further, we think that our approach can be extended with a more general shader markup language. Currently, we only support the markup of semantic properties and visual attributes. A more general solution would allow us to tag resources (such as volumes and textures) and parameters that are (or shall be) exposed in the user interface. With the extension of the shader markup language, a more general visualization system could be realized. The markup of shader files would be sufficient for the rapid generation and exploration of new semantics-driven visualization systems.

3.8 ACKNOWLEDGMENTS

We thank the participants of our user evaluations for sharing their time and expertise. This work has been partially funded by the ViMaL project supported by the Austrian Science Fund (FWF), grant no. P21695. The MRI head dataset used for generating Fig. 3.14 and Fig. 3.15 is courtesy of the BCN Neuroimaging Center, Groningen.

This chapter is based on a previously published article [Gerl et al., 2012]. Moritz Gerl, Peter Rautek, Tobias Isenberg, and M. Eduard Gröller. Semantics by Analogy for Illustrative Volume Visualization. In *Computers & Graphics*, 36(2):201–213, 2012.

4

FROM INTERACTIVE TO SEMI-AUTOMATIC CONTROL OVER THE RESULT

Using a graphical rule-specification interface to define mappings of semantic properties to visual attributes as described in the previous chapter improves upon the usability and flexibility of semantics-driven volume rendering. The semantics-by-analogy system provides the user with a flexible, direct, and intuitive control over the result. However, the mapping specification discussed in Chapter 3 requires a selection of the semantic properties on which the mapping is based. This selection of input and output parameters is also intrinsically necessary for the original semantic-layers technique by Rautek et al. [2007, 2008a] where the input and output of the visualization mapping forms the basis for the visualization rules. In other words, whenever the user of a semantics-driven volume rendering system formulates a visualization rule, for example *'if density is high then color is green,'* the user has to come up with the input (*'density'*) and the output (*'green'*) properties of the mapping. In Chapter 3 we illustrate the usage of a variety of different semantic properties such as *z-coordinate, object-space position, xz-gradient, diffuse illumination, distance along the ray, etc.* in illustrative visualization mappings. The concept of semantic shader augmentation is a highly flexible manner of specifying the input and output of the visualization mapping, as emphasized repeatedly. However, the concept of semantic shader augmentation comes with the downside that the input and output to the desired mapping *have to* be explicitly defined by a visualization expert to begin with. Once this definition has been performed initially, the semantics-by-analogy system can be used by a domain expert or illustrator. The selection and definition of semantic properties, however, *has* to be performed in any case. It is an unavoidable prerequisite of the method presented in Chapter 3. To alleviate this problem, we explore a different way of controlling the input to the rendering function in Chapter 5. We take the specification of the input to the function out of the hands of the user again and employ machine learning to automatically learn an adequate mapping. Furthermore, we also restrict the output of the rendering function to a particular visual abstraction, which is pen-and-ink hatching. We then provide direct interaction tools to let the user control the application of this static input and output. We examine this way of realizing control over the result in the development of a method for interactive example-based pen-and-ink hatching.

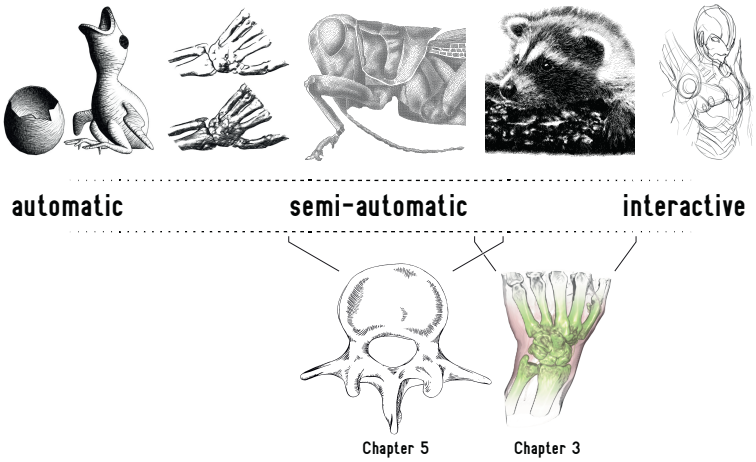


Figure 4.1: The location of the two core parts of this thesis in the interaction continuum introduced in Chapter 2. Both semi-automatic methods cover a sub-spectrum of the interaction continuum. The semantics-driven volume illustration method discussed in Chapter 3 is more toward the interactive side of the continuum, while the example-based hatching method presented in Chapter 5 uses more automated control over the result.

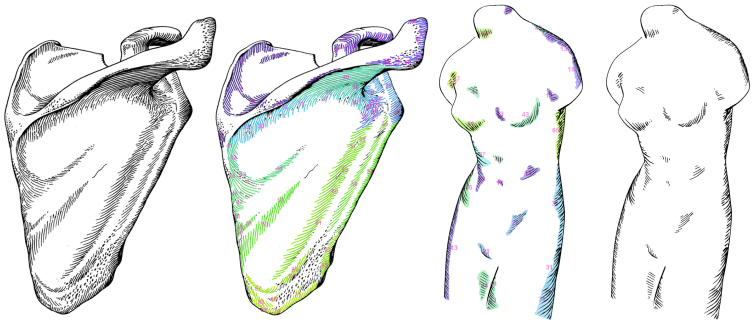
Let us position our two different methods within the interaction continuum that we introduced in Chapter 2. The approximate location of the two methods in our interaction continuum is indicated in Fig. 4.1. Both semi-automatic methods are located toward the interactive side of the center of the interaction continuum. Both methods can be employed with a varying degree of user interaction. Therefore, the two methods each cover a sub-spectrum of the interaction continuum. The semantics-driven volume illustration method discussed in Chapter 3 covers a sub-spectrum which is located more toward the interactive side of the continuum due to the flexibility of the involved user interactions. This positioning might appear confusing at first glance because the rendering part of the volume illustration method in Chapter 3 is fully automatic. The control over the result, however, is in the hands of the user to a large degree. This is why we position the illustrative visualization method more toward the interactive side of the spectrum. The example-based hatching method presented in Chapter 5 uses more automatic control over the result. We could as well position the hatching method in Chapter 5 more toward the automatic side of the interaction continuum and illustrate it to cover the entire sub-continuum from automatic to semi-automatic, because the hatching method can as well be used fully automatically. The sub-spectrum that we indicated in Fig. 4.1, however, refers to the semi-automatic usage of the hatching system.

The hatching method for 3D surface meshes described in Chapter 5 examines the previously outlined semi-automatic way of implementing control over the result. In this method, we select a fixed set of surface features that forms the input to the visual mapping that is performed by the hatching renderer. The set of selected surface features is somewhat similar to the semantic properties that we use in Chapter 3. The surface features we employ in our example-based hatching method comprise spatial, directional, and lighting properties such as *image-space coordinates, depth, facing-ratio gradient, curvature direction, diffuse illumination, etc.* In contrast to the manual mapping specification in Chapter 3, however, we use machine learning in Chapter 5 to automatically learn a mapping of this fixed set of surface features to hatching properties. The hatching properties are the hatching regions, the types of hatching strokes, the hatching directions and distances between the strokes, as well as low-level properties such as the stroke width and shape. The example-based mapping from surface features to hatching properties is established once in a separate learning stage. The learned mapping is then applied in a semi-automatic synthesis stage. In the synthesis stage, the user of the hatching system described in the following chapter has no control over *which* properties are decisive for the resulting rendering. This is a major deviation from the dynamic mapping adjustment in Chapter 3, where the decisive features can be changed at any time. The user of the hatching system in Chapter 5 does also have no choice of the visual abstraction that is generated by the renderer. The renderer does always create hatching illustrations. This is another major difference between the two core parts of this thesis. The semantics-driven volume renderer in Chapter 3 is a flexible framework capable of realizing a wide variety of visualization mappings but requires expert knowledge to come up with a suitable mapping to begin with. The hatching renderer in Chapter 5, in contrast, is tailored to the generation of pen-and-ink hatching illustrations. The hatching method uses a fixed set of surface features and employs machine learning to learn a mapping of these surface features to hatching properties from hand-drawn examples. The learned mapping can then be applied semi-automatically by the user. Although the input to the rendering function (i. e., the set of surface features) as well as the generated visual abstraction are statically pre-defined in the example-based hatching system described in Chapter 5, the user of this system still has many options for adjusting the rendering result. Interaction with the result is provided on a higher level, as outlined before in Chapter 2. The most important of these user interactions is the brushing with patches of hatching strokes onto a 3D model. This brushing interaction gives the user full control over where to place which kind of strokes. The strokes that are automatically generated in the brushed regions incorporate the learned lower-level drawing characteristics.

Together, the described semi-automatic example-based control over the result combines the advantages of both interactive and example-based illustrative rendering approaches. This control gives artists and illustrators the creative freedom and editability they need while at the same time it provides the computer assistance that is necessary to rapidly create hatching illustrations of a reasonable aesthetic quality. Furthermore, the example-based interactive control gives people who are not proficient in fine arts and illustration the possibility to interactively create pen-and-ink hatching illustrations of 3D models.

5

INTERACTIVE EXAMPLE-BASED PEN-AND-INK HATCHING



ABSTRACT: *This chapter presents an approach for interactively generating pen-and-ink hatching renderings based on hand-drawn examples. We aim to overcome the regular and synthetic appearance of the results of existing methods by incorporating human virtuosity and illustration skills in the computer generation of such imagery. To achieve this goal, we propose to integrate an automatic style transfer with user interactions. This approach leverages the potential of example-based hatching while giving users the control and creative freedom to enhance the aesthetic appearance of the results. Using a scanned-in hatching illustration as input, we use image processing and machine learning methods to learn a model of the drawing style in the example illustration. We then apply this model to semi-automatically synthesize hatching illustrations of 3D meshes in the learned drawing style. In the learning stage, we first establish an analytical description of the hand-drawn example illustration using image processing. A 3D scene registered with the example drawing allows us to infer object-space information related to the 2D drawing elements. We employ an hierarchical style transfer model which enables us to capture drawing characteristics on four levels of abstraction, which are global, patch, stroke, and pixel levels. In the synthesis stage, an explicit representation of hatching strokes and patches of strokes enables us to synthesize the learned hierarchical drawing characteristics. Our representation makes it possible to directly and intuitively interact with the hatching illustration. Amongst other interactions, users of our system can brush with patches of hatching strokes onto a 3D mesh. This interaction capability allows artists and illustrators who are working with our system to make use of their virtuosity and illustration skills while interactively creating hatching illustrations of 3D meshes. This interaction capability improves upon the aesthetic quality and illustration effectiveness of the automatic results that are achievable with our approach. Furthermore, the proposed interactions allow people without a background in hatching to interactively generate visually appealing hatching illustrations.*

5.1 INTRODUCTION

The computer generation of pen-and-ink hatching illustrations has a long tradition in non-photorealistic and illustrative rendering. But although pen-and-ink rendering methods have been introduced quite some time ago, there seems to be little adoption outside the field. This contrasts the long tradition and widespread use of handcrafted pen-and-ink illustrations. Over centuries, those depictions have been proven to be an effective means of visually communicating information. And they are still widely used nowadays, e. g., in medical training. We believe that the lack of adoption of computer-generated hatching illustrations can at least in part be explained by their synthetic and overly regular visual appearance. A comparative study of Isenberg et al. [2006] has, amongst other findings, shown that computer-generated illustrations are clearly distinguishable from hand-drawn illustrations due to a lack of 'character'. We aim at incorporating human virtuosity and illustration skills into the computer generation of pen-and-ink hatching illustrations to improve upon the aesthetic appeal and illustration effectiveness of such imagery. Recently, Kalogerakis et al. [2012] presented an approach that shares our goals. In this chapter we present an approach that uses an explicit representation of drawing elements, in contrast to the pixel-based approach chosen by Kalogerakis et al. [2012]. This representation allows us to transfer drawing characteristics on higher levels than a pixel grid and to provide user interactions for adjusting the resulting illustrations. We present a semi-automatic strategy for implementing control over the result in illustrative rendering in this chapter. The strategy is to combine automatic control via machine learning with interactive control via user adjustments. Our approach provides the following contributions:

IMAGE ANALYSIS ON HATCHING DRAWINGS: We apply image processing methods to the detection of the stroke trajectories in a hand-drawn and scanned-in pen-and-ink hatching illustration. This image analysis allows us to learn the drawing style from a given example image as well as to use the example image for texture mapping.

3D INFORMATION FOR EXISTING 2D DRAWINGS: We employ a 3D scene registered with a hand-drawn example illustration to infer 3D information related to the 2D elements in the example. This enables us to include 3D measurements in learning a model of the example drawing style.

ANALYTICAL REPRESENTATION OF DRAWING ELEMENTS: We represent patches of strokes and stroke trajectories analytically in object-space. This representation allows us to transfer drawing characteristics on a stroke level (as opposed to the transfer of hatching properties on a pixel level). The analytical representation of hatching patches and hatching strokes also facilitates user interactions for controlling the result.

ADAPTIVE SURFACE PATCHES: We introduce the use of a real-time example-based mesh segmentation to create adaptive surface patches as the basis for creating patches of hatching strokes. These example-based surface patches allow us to capture and reproduce global characteristics of an example illustration. While we use the surface patches for generating hatching strokes, the notion of patch-wise style transfer can be generalized to other depiction styles.

STROKE DISTANCE FUNCTIONS: We propose to learn the interrelationship of hatching strokes locally as a function of object-space mesh features. This allows us to model the stylistic variations of the example illustration style on a stroke level. This local model of stroke distances results in hatching patterns that are less uniform and less regular than the hatching patterns generated by previous approaches.

INTERACTION CAPABILITIES: We provide users of our system with the possibility to interact with the resulting illustration. Users of our system can alter the appearance and direction of hatching strokes within individual patches of strokes, can flexibly retouch the stroke trajectories, and brush with patches of hatching strokes. These direct interactions with the hatching illustration are an effective means of combining the benefits of automatic style transfer with human creativity and virtuosity, which enables an application of our method in creative environments. Furthermore, the interaction capabilities of our approach allow users without a background in hatching to interactively create visually appealing hatching illustrations in a short amount of time.

This chapter is structured as follows. We first review related work in Section 5.2. Next, we give an overview of our approach in Section 5.3. We then detail how we capture a drawing style in Section 5.4 and explain how we reproduce the learned style in Section 5.5. We explain our user interactions in Section 5.6. In Section 5.7 we present some results of our method and discuss its limitations in Section 5.8. We conclude the chapter and describe possibilities for future work in Section 5.9.

5.2 RELATED WORK

Many different ways for creating hatching renderings have been explored in the past. Saito and Takahashi [1990] introduce the usage of isoparametric lines to create hatching images. This idea was further developed by other researchers [Elber, 1995; Winkenbach and Salesin, 1996]. Already in this first generation of hatching techniques, Winkenbach and Salesin [1994] incorporate concepts derived from hand-drawn hatching illustrations, such as the notion of varying the width along a stroke in order to simulate marks which are created by applying ink to paper with a nib pen. Girshick et al. [2000] introduce the creation of strokes based on

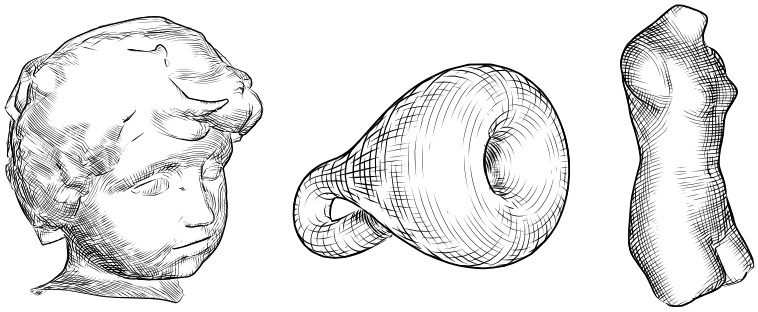


Figure 5.1: Smooth direction field hatching by Hertzmann and Zorin [2000].

principal curvature directions, a notion used by many following hatching techniques. Hertzmann and Zorin [2000] perform an optimization of the curvature directions and use the resulting smooth direction field to create hatching strokes (see Fig. 5.1). In our work, we use this optimized direction field as a basis for learning the directions of example hatching strokes. Zander et al. [2004] propose to create object-space hatching strokes by tracing the curvature directions in object space (see Fig. 5.2). In our approach, we also use an object-space representation of the stroke trajectories. In contrast to these explicit stroke descriptions, Praun et al. [2001] introduce a hatching approach using textures. This allows for real-time rendering, while still achieving expressive results (see Fig. 2.1). Praun et al.'s [2001] technique is extended by Kim et al. [2008] to work for dynamic and specular surfaces. Gasteiger et al. [2008] build upon Praun et al.'s [2001] technique in order to generate hatchings that are oriented along model-based preferential directions of anatomical surfaces. Despite the appealing properties of these texture-based methods, we decided to use an explicit stroke representation. This representation gives us the required control over singular strokes which we need for reproducing learned stroke properties. All the mentioned methods have in common that the hatching strokes depend on a mapping of a small set of lighting conditions and mesh features to stroke properties. It is thus very difficult for these methods to convincingly reproduce the stylistic properties and variations present in hand-drawn hatchings. For this reason, Kalogerakis et al. [2012] recently proposed to learn hatching styles from example drawings (see Fig. 5.3). While Kalogerakis et al.'s [2012] work serves as an inspiration of our own, we improve on it by using analytical representations of hatching patches and hatching strokes, by modeling stroke distance interrelationships in more detail, as well as by specifically permitting interaction. We explain how we deviate from Kalogerakis et al.'s [2012] work in more detail below.

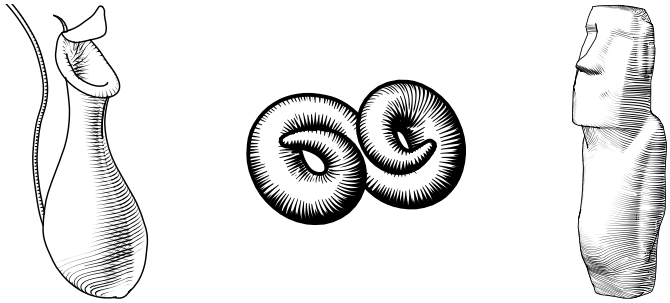


Figure 5.2: Object-space hatching by Zander et al. [2004].

There has been some previous effort in style transfer and example-based illustrative rendering. Hamel and Strothotte [1999] capture user-defined rendering parameters and re-use them for rendering other meshes. Mertens et al. [2006] transfer texture variations between meshes by correlating texture properties with geometric mesh features, using similar features as we do in our approach. Hertzmann et al. [2001] present a framework capable of learning and reproducing image processing filters which mimic drawing and painting styles, taking 2D images as input. Although many different styles can be successfully transferred, the pixel-based nature of this technique makes it difficult to faithfully reproduce drawing styles which depend on long and individual strokes. Zhao and Zhu [2011] generate example-based portrait paintings from photographs by transferring brush strokes from a collection of template paintings. Also working on 2D images, Kim et al. [2009] adopt statistical texture transfer methods to transfer the stippling characteristics from example stippling illustrations to new images (see Fig. 2.6a). Related to this, Martín et al. [2011] present a method for scale-dependent and example-based stippling based on halftoning (see Fig. 2.6b). Stippling by example is mostly concerned with calculating adequate stipple positions and shapes. Herein, the individual stipple points do not have a function on their own, but work as a conglomerate. In the drawing style which we aim to reproduce, in contrast, each drawing mark has an individual function. Furthermore, the hatching strokes are subject to interrelations along their entire extent, as opposed to the interrelations of only one 2D location per mark in the case of stippling. For these reasons, hatching by example requires more complex style transfer models than required by example-based stippling.

Some approaches establish statistical models of stroke patterns. Jodoin et al. [2002] as well as Barla et al. [2006] synthesize stroke patterns by example based on statistics which they derive from given input stroke patterns. Other related work [Freeman et al., 1999, 2003; Hertzmann et al., 2002; Kalnins et al., 2002] focuses on the style transfer between curves.

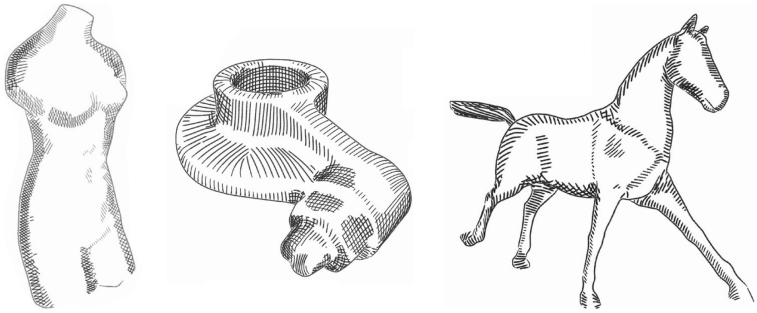


Figure 5.3: Example-based hatching in image space by Kalogerakis et al. [2012].

These approaches transfer stroke patterns or line rendering styles merely in 2D. For our needs, however, it is necessary to involve 3D information in the style transfer process. For this reason, we condition our style transfer model on a 3D object, and also apply the style transfer model to the generation of renderings of 3D objects. This is inspired by the work of Lum and Ma [2005] as well as that of Cole et al. [2008]. Both of these approaches use machine learning to correlate hand-drawn line drawings with computer-generated silhouettes and feature lines.

Applying machine learning to learn hatching properties was only recently introduced by Kalogerakis et al. [2012](see Fig. 5.3). Their approach operates on pixels. Both the learning and the synthesis of a drawing style are performed on a pixel basis. Although Kalogerakis et al. [2012] synthesize the final hatching strokes as textured triangle strips, they learn and infer hatching properties per pixel. The transfer of drawing characteristics based on pixels means that the transfer does not involve any explicit representation of drawing elements, such as strokes. The results of Kalogerakis et al. [2012] prove that this strategy works very well for the illustration styles they work with. For learning the illustration styles that we aim to reproduce, however, we need different learning strategies, in particular a different drawing representation. Therefore, we operate on explicit analytical representations of hatching strokes and patches of strokes. This explicit representation of drawing elements enables us to capture drawing characteristics and stylistic properties present in four nested levels of abstraction. These levels are a global, patch, stroke, and pixel level. Representing the hatching strokes explicitly *during* the style transfer process allows us to transfer local stroke distance characteristics, which results in less uniform and equidistant strokes than Kalogerakis et al.'s [2012] global pixel-based approach to transferring stroke distances. Furthermore, the method proposed by Kalogerakis et al. [2012] delivers a static result which cannot be modified after its generation. Our explicit description of drawing elements, in contrast, opens up the possibility to

Figure 5.4: Interactively generated pen-and-ink illustration by Salisbury et al. [1997].

interactively modify the resulting illustration. We exploit this control by providing users of our system with the possibility to brush with patches of hatching strokes onto a 3D model, amongst other interactions. The interaction capabilities of the resulting semi-automatic hatching system allow users of our system to manually enhance the aesthetic quality of the results. This possibility for adjusting the result gives our method the potential of being employed in creative environments as well as by laymen in hatching illustration. Moreover, Kalogerakis et al. [2012] use an extensive set of surface features in their style transfer model. This vast feature space makes their model very accurate but also computationally expensive. This computational load results in rather long processing times in the order of 30 to 60 minutes for the image synthesis. To avoid this delay we use a much smaller set of features. This makes our model less accurate than that of Kalogerakis et al. [2012], but brings us close to the possibility of responsive interaction with the example-based illustration.

The concept of interactive illustrative rendering has been proven successful, not only since Seims [1999] advocated to provide more user control for fully automatic non-photorealistic rendering methods. Salisbury et al. [1994, 1997] let users brush with stroke patterns to interactively create pen-and-ink illustrations (see Fig. 5.4). Deussen et al. [2000] employ user-defined segmentation images and brushing interactions for the semi-automatic generation of stippling drawings (see Fig. 2.5). Another semi-automatic illustration system is presented by Ostromoukhov [1999], who employs user-defined 2D patches as the basis for generating digital facial engravings using 2D images as reference images. Rössl and Kobbelt [2000] also use image-space segmentations in their interactive system for creating line-art renderings. Here, the segmentations

are created automatically, but can be adjusted by the user. We also rely on a user-adjustable automatic segmentation to identify regions of different drawings characteristics. However, we propose to perform the segmentation on the mesh and to learn the segmentation using machine learning methods, similar to the approach of Kalogerakis et al. [2010] for learning mesh segmentation and labeling. Furthermore, we provide users with the possibility to interactively modify the resulting illustration by allowing them to refine the mesh segmentation. This interaction is related in its intention to the direct tweaking of lighting and shading as proposed by Anjyo et al. [2006] and extended by Todo et al. [2007]. We do not discuss the field of mesh segmentation in detail in this thesis. The interested reader might consult the following references. The survey of Shamir [2008] compares different mesh segmentation techniques. Chen et al. [2009] propose a benchmark for mesh segmentation and discuss the performance of different methods for mesh segmentation.

Furthermore, Breslav et al. [2007] also use patches embedded on the surface for creating illustrative renderings. They use pre-defined 3D patches to transform 2D patterns in a way that the transformed 2D patterns match the underlying 3D transformation. In contrast to pre-defined patches, we use adaptive 3D patches that are dynamically predicted based on a learned function of lighting conditions and mesh features. And instead of transforming 2D patterns, we use the 3D patches to guide the generation of stroke trajectories in 3D.

5.3 OVERVIEW

Our overall approach consists of two general stages and is illustrated in Fig. 5.5. First, we learn a model of an illustrator's pen-and-ink hatching style (Section 5.4). Then, we apply this model to synthesize hatching illustrations of target 3D meshes (Section 5.5). The synthesis can be influenced by user interactions (Section 5.6).

As input to the learning stage (see Fig. 5.6) we use a hand-drawn hatching illustration, a manually created segmentation image of this illustration, and a 3D scene whose projection closely matches the example image. We refer to this 3D replication of the example drawing as 'registered 3D model'. We use it to infer 3D information related to the 2D example image. We obtain it manually by sculpting it from a 3D model similar to the object depicted in the illustration, or by using illustrations that were drawn using 3D renderings as master [Isenberg et al., 2006].

In a preprocessing step to the learning procedures, we employ image processing methods to detect the trajectories of the hatching strokes in the example image (Section 5.4.1). This stroke detection is facilitated by

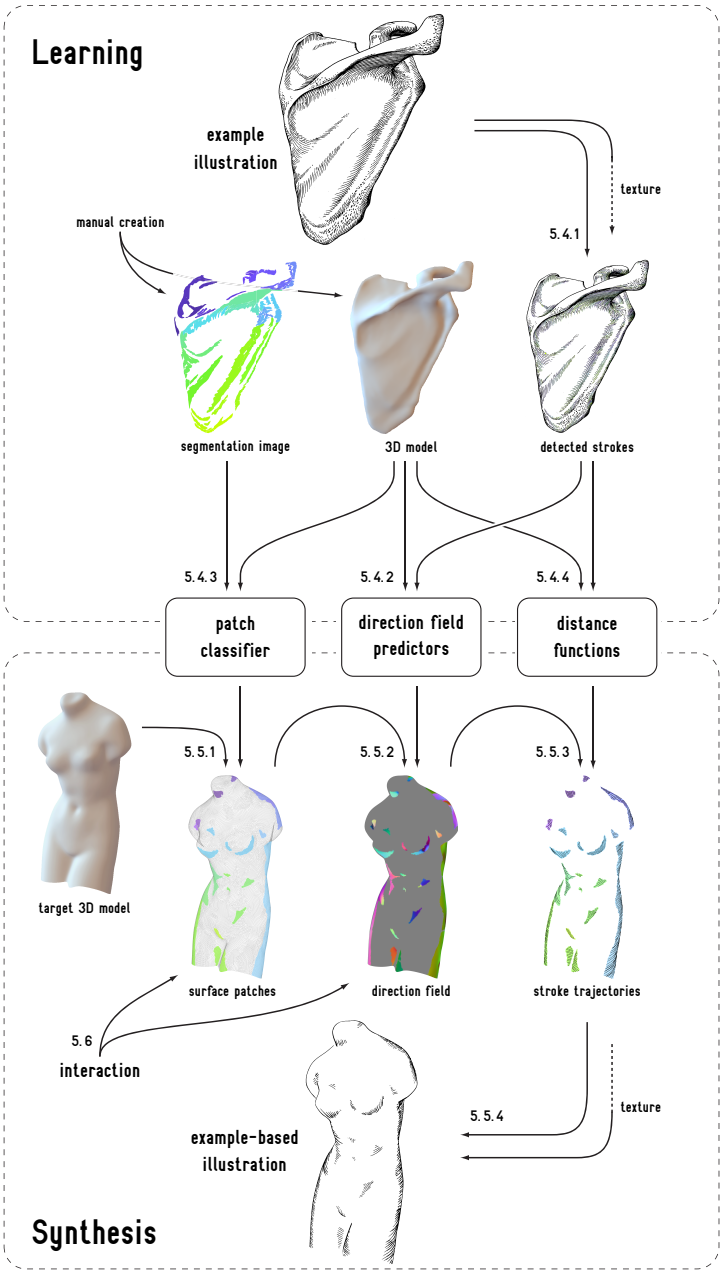


Figure 5.5: Overview of our approach to hatching by example. In a learning stage, we learn the characteristics of an illustrator's hatching style. In a synthesis stage, we apply the learned style to a target 3D model to gain an example-based illustration. The arrows are annotated with the section numbers where the corresponding processes are explained.

using the manually created segmentation image to separate groups of strokes from the remainder of the input image.

The segmentation image also defines patches of strokes in the example illustration which function as a group and which share common properties. We attempt to learn the properties of these groups of strokes in the following way. Using the registered 3D model and the segmentation image, we train a classifier that classifies the vertices of the input mesh into segment labels based on lighting and geometric mesh features (Section 5.4.2). When we apply this classifier on a target 3D mesh in the synthesis stage, it yields a dynamic segmentation of the mesh into surface patches which incorporate the learned global drawing characteristics (Section 5.5.1). The patch classifier operates on the whole mesh, and represents the first level of our four-level hierarchy of hatching style descriptors. To complement the described automatic inference of hatching regions, the surface patches can also be interactively modified by the user via brushing (Section 5.6).

The second level of our hierarchy is concerned with the directions of the hatching strokes. We learn the properties of the stroke directions in the example illustration on a patch level. To be able to do this in 3D , we use the following approach: we reproject the detected 2D stroke trajectories onto the registered 3D model. In this way, we gain an object-space description of the trajectories of the hand-drawn strokes. We then use regression analysis to learn how the 3D stroke trajectories correlate with lighting and geometric features of the registered 3D model (Section 5.4.3). Note that the reprojected strokes allow us to learn the directions of the strokes *on the surface*, rather than learning the *image-space* directions of strokes in dependency of surface features. We train one regression function for each patch of strokes in the example image. In the synthesis stage, every surface patch is assigned one example stroke patch. We then apply one direction field function per surface patch to infer a direction field which incorporates the stroke directions learned from the corresponding example stroke patch (Section 5.5.2).

The third level of our model deals with the distance relationship of individual strokes with their neighboring strokes. Here we use the same reprojection setup as described above. The reprojection setup allows us to correlate 2D stroke distances with 3D lighting conditions and surface features (Section 5.4.4). In order to establish this correlation, we train a regression function for each individual stroke in the example illustration. In this way, we learn the 2D distances of the stroke to its neighboring stroke along its extent as a function of the surface features measured at the locations of the reprojected stroke control points. In the synthesis stage, we use these stroke distance functions during the tracing of stroke trajectories to reproduce the recorded patterns of stroke interrelationship (Section 5.5.3). This *local* approach to learning stroke distances and the

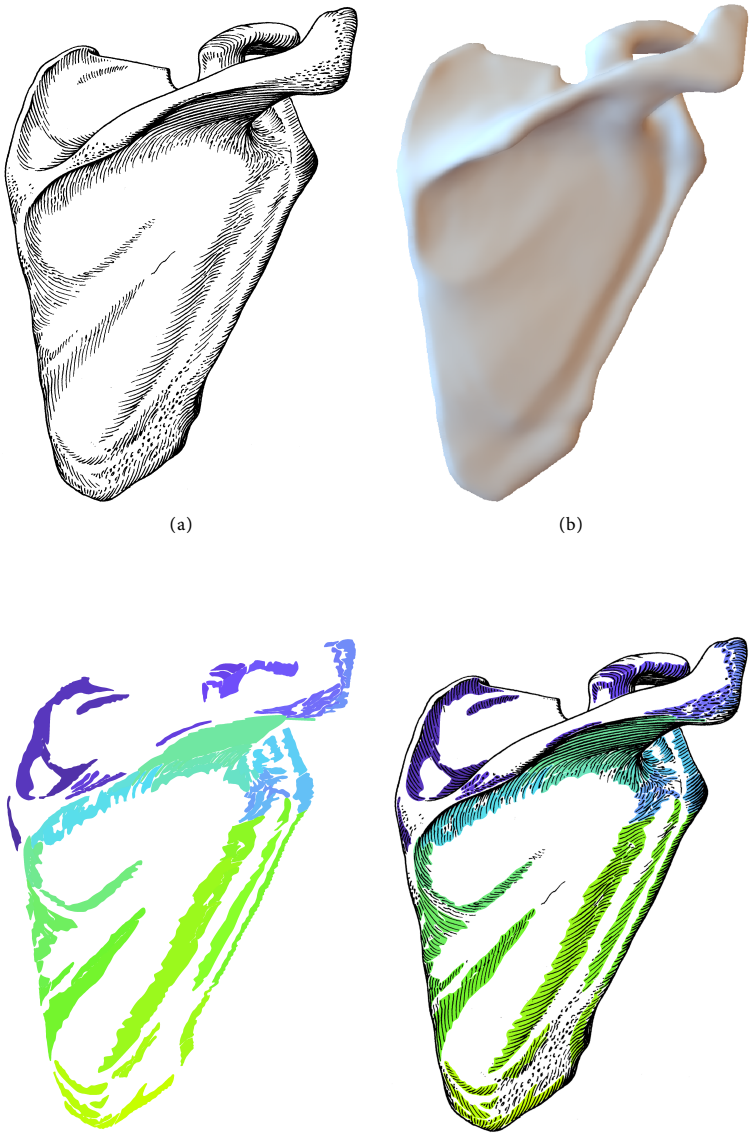


Figure 5.6: The input to our approach for learning a hatching style. (a) A hand-drawn hatching illustration (courtesy of Dauber et al. [2005]), (b) a 3D scene whose projection closely matches the example image, and (c) a manually created segmentation image of the example illustration ((d) shows an overlay of the latter two images for reference).

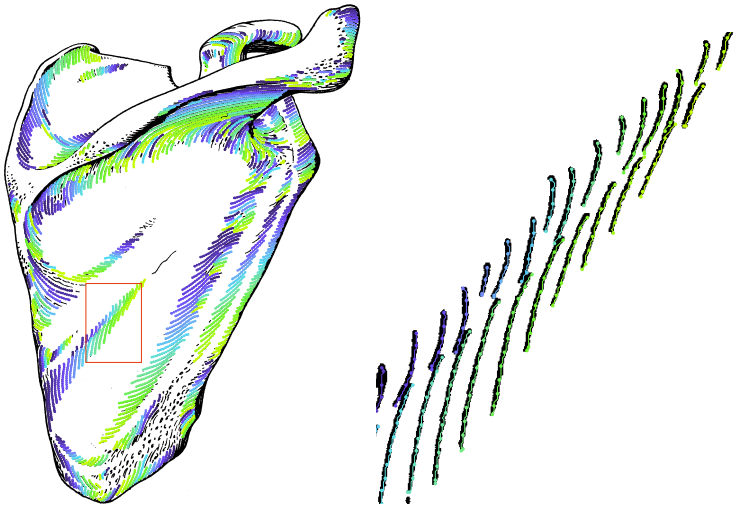
explicit representation of strokes *during* the style transfer process allows us to transfer stroke distance characteristics in more detail than the *global* pixel-based approach of Kalogerakis et al. [2012]. Our local approach to modeling stroke distances results in hatching strokes that are less regular and less uniform and arguably exhibit more 'character.'

The final step in the synthesis is to create 2D textured triangle strips from the 3D stroke trajectories (Section 5.5.4). This stroke rendering represents the fourth level of our style descriptor hierarchy and involves two attempts of transferring low-level stroke properties.

The resulting hatching illustration can be manually refined by mapping and brushing operations (Section 5.6). This semi-automatic system effectively combines the advantages of automatic example-based hatching and human virtuosity. This interaction capability can increase the aesthetic quality and illustration effectiveness of the results that are achievable with our method. First, the illustration can be altered by overriding the learned mapping of surface patches to example stroke patches. By reassigning a different example stroke patch to a surface patch, the strokes within this patch can be changed. Second, the hatching angle of each individual patch can be controlled. Third, the direction field that we use as a reference for inferring the stroke direction field can be retouched with brushing tools. Fourth, we allow users of our system to brush with patches of example-based hatching strokes. All generated hatching strokes depend on the dynamic mesh segmentation (Section 5.5.1). Users can easily modify, add, or remove patches of strokes by refining this segmentation with a set of brushing interactions. Thus the interaction also allows us to address some of the limitations of the fully automatic approach (Section 5.8).

5.4 LEARNING A HATCHING STYLE

In this section we explain the learning part of our approach in more detail. The illustration style that we aim to learn is a specific type of traditional pen-and-ink hatching. Fig. 5.6a shows an example from an anatomy textbook [Dauber et al., 2005]. A property of this hatching style is that the drawing is, for the most part, composed of separate individual strokes. Each stroke in the drawing has a particular function. This contrasts other, more areal, hatching styles which use many overlapping strokes (styles that are visually similar to, e. g., the real-time hatching images of Praun et al. [2001]). Based on this property we can automatically detect the strokes in hand-drawn images that are drawn in our target illustration style. The detection of strokes would be harder to accomplish on an image consisting of many overlapping strokes.



(a) (b)

Figure 5.7: Stroke detection. We detect trajectories of hatching strokes in a given example illustration using image processing ((b) shows a detail of (a)).

5.4.1 *Image Analysis*

By establishing an explicit analytical representation of the strokes in an example illustration, we create the possibility to learn the properties of the strokes on higher levels than a mere pixel representation would allow us to do. As input to our stroke detection, we use a high-resolution black-and-white scan of an example drawing (Fig. 5.6a). As second input, a manually created segmentation image (Fig. 5.6c) allows us to separate patches of strokes from the rest of the drawing. It also helps us to exclude individual strokes that we cannot reproduce faithfully, e. g., the stippling marks in the lower right part of the shoulder blade illustration. For each patch, we run a series of standard morphological operations [Soille, 2003] to detect the trajectories of the hatching strokes. Our image processing pipeline starts with a morphological cleaning operation (an operation that removes isolated pixels) to remove scanning artifacts. Then we use connected component labeling to identify the strokes. Thinning the stroke regions yields skeletons of the strokes. A hit-or-miss transform identifies skeleton junctions and endpoints which we use to prune the skeletons and to separate the trajectories of overlapping strokes from each other. We then vectorize the strokes by creating equidistant control points along the stroke skeletons. Fig. 5.7a shows the result of the described stroke detection routines, Fig. 5.7b shows a detail section. We used Matlab® to implement the stroke detection routines. We employ

the following Matlab® commands for morphological cleaning, connected component labeling, thinning, as well as for the detection of skeleton endpoints and junctions:

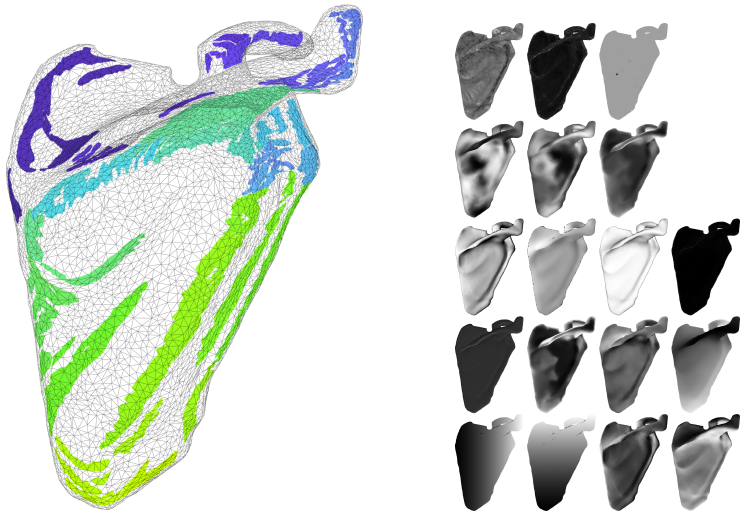
```
Source = bwmorph(Source,'clean',Inf);
[Boundaries, Labels] = bwboundaries(Source,'noholes');
Skeletons = bwmorph(Source,'thin',Inf);
Endpoints = BOHitOrMiss(Skeletons,'end');
Junctions = BOHitOrMiss(Skeletons,'triple');
```

This sequence of commands is executed for each separated patch of example strokes after using the segmentation image to separate a patch of strokes from the remainder of the example image. After running the given routines, we use the number of endpoints and junctions to differentiate between different stroke types. For stroke types consisting of overlapping strokes, we discriminate the overlapping strokes from each other by starting at the longest skeleton segments and searching for adjacent shorter skeleton segments that are oriented in the same direction (at an angle smaller than 45°) as the long segments. In this way we resolve ambiguities at stroke crossings and gather the skeleton segments that belong to the same stroke. We omit small strokes whose area is below 5% of the average stroke region within a patch or whose skeleton length is below 50 pixels. During vectorizing the stroke trajectories, we generate control points in a distance of 25 pixels along the stroke skeletons for an input image of 3670×7360 pixels (Fig. 5.23a).

By reprojecting the detected stroke trajectories onto the registered 3D model, we can relate object-space properties of the hand-drawn strokes to 3D measurements. This *reprojection* of drawing elements to object-space is a novelty compared to previous approaches that also use a 3D model registered with a 2D drawing, but merely *read out* 3D information related to 2D elements in an input drawing [Kalogerakis et al., 2012; Lum and Ma, 2005; Cole et al., 2008]. Our approach, in contrast, allows us to learn *object-space* properties of otherwise image-space elements, such as the directions of hatching strokes *on the surface*. Before we do that on a patch level and locally, however, we make an attempt to capture global properties of the drawing style as outlined next.

5.4.2 *Patch Properties and Surface Features*

We identify the grouping of strokes in patches as a central stylistic element of our target illustration style. It can be seen in Fig. 5.6d that the strokes in each patch share common attributes such as direction, width, and shape. In order to take this grouping of strokes into account, we explicitly involve it in our style transfer model. This explicit handling of stroke groups is an essential distinction between our approach and the global pixel-based approach of Kalogerakis et al. [2012]. The manual seg-



(a) (b)

Figure 5.8: Learning patch properties. We use (a) the segmentation image and the registered 3D model to learn the global properties of patches of strokes with respect to (b) lighting and geometric mesh features as described in the text.

mentation of the example illustration allows us to faithfully distinguish the different patches of similar strokes. This would be difficult to realize automatically, as it involves complex perceptual and creative decisions. We use the segmentation image together with the registered 3D model (Fig. 5.8a), which we denote here as the input mesh, for capturing the properties of these groups of strokes. We project each vertex of the input mesh to image-space and read out the patch label found at this location in the segmentation image. Assigning a patch label to each vertex represents a segmentation of the input mesh that matches the segmentation given by the segmentation image. We take this mesh segmentation and a number of lighting and geometric features (as detailed below) measured at each vertex and train a classifier to learn a mapping from mesh features to segment labels. Applying this classifier in the synthesis stage assigns a segment label to each vertex of a target mesh (Section 5.5.1). The resulting dynamic segmentation of the target mesh into surface patches incorporates the learned global characteristics of the example illustration as defined by the segmentation image.

We use a voting multiclass classifier [Hastie and Tibshirani, 1998] with a one-vs.-one strategy for classification. A multiclass classifier is a classifier that distinguishes between n classes, as opposed to a binary classifier that distinguishes between two classes. A voting multiclass classifier is a combination of multiple binary classifiers. We employ relevance vector

machines [Tipping and Faul, 2003] with radial basis function kernels as binary classifiers. We experimented with various classifiers and gained the most promising results with the named one. We use a stopping epsilon of $\varepsilon = 0.001$ for the multiclass classifier. In all of our learning routines, we use a gamma of $\gamma = 0.08$ for the radial basis function kernels.

As mentioned before, we use a relatively small set of surface features compared to the approach of Kalogerakis et al. [2012]. We selected a set of 18 decisive features. We identified these by correspondence with professional artists and illustrators, by drawing conclusions from the literature on computer-generated hatching, and by experiment. From artists and from the literature [Hodges, 2003]we learned that lighting features are most influential on the decision on where to place strokes. An illustrator did let us know that directional features, such as the curvature direction, are often used for conveying shape, but that the direction of the strokes does not necessarily have to follow the curvature direction. We experimented with various features, and selected a set of features that lead to a robust classification of patch properties. We consider a classification as robust if it results in continuous patches which, assessed by subjective reasoning, match the areas used by the creator of the example image. We made a tradeoff of classification speed and accuracy for choosing the number of features. The classifier operates on feature vectors of scalar values. For including 2D and 3D measurements in our model, we either use their components or the dot product with the view vector as a view-dependent scalar of a 3D variable. Fig. 5.8b depicts renderings of the employed features in reading order and as listed below.

The six view-independent features we use are: the first and second principal curvature magnitudes $|\kappa_1|$ and $|\kappa_2|$, the 'parabolicalness' $|\kappa_1|/|\kappa_2|$, as well as the x-, y- and z-components of the first principal curvature direction after performing the curvature optimization procedure proposed by Hertzmann and Zorin [2000] λ_{1_x}, λ_{1_y}, and λ_{1_z}, which we here denote as the first optimized curvature direction.

The 12 view-dependent features we use are: diffuse illumination I (Lambertian shading), approximated global illumination SSDO (screen-space directional occlusion as introduced by Ritschel et al. [2009]), facing ratio $n \cdot v$ (where n is the normal and v is the viewing direction), facing ratio gradient magnitude $|\nabla(n \cdot v)|$, view-dependent facing ratio gradient direction $(\nabla(n \cdot v)) \cdot v$, view-dependent first optimized curvature direction $\lambda_1 \cdot v$ (where λ_1 is the first optimized curvature direction), view-dependent second optimized curvature direction $\lambda_2 \cdot v$, depth z, the image-space coordinates x_i and y_i, as well as the x- and y-components of the normal projected to image space n_{i_x} and n_{i_y}.

We normalize and weight the features to control the influence of each of the features individually. The weighting is achieved by multiplying each feature with a user-controllable weight. Multiplying a feature that is

normalized to the range of [0,1] with a weight greater than 1 causes the scaled feature to have greater impact during learning and inference.

We put most emphasis on the lighting features as we assume them to have the greatest impact on where the illustrator has drawn strokes and on which kind of strokes he or she used in different regions. We see this assumption confirmed by the literature on illustration [Hodges, 2003]. The assumption is also reflected by the ranking of features reported by Kalogerakis et al. [2012]. We weight the image-space features slightly stronger than the directional features. When we learn a model of the stroke directions and distances, we adjust the feature weights accordingly. The feature weights we use are given in Table 1, listed in the same order as the features are named above.

Feature	Regions	Directions	Distances
$\lvert\kappa_1\rvert$	1.0	2.0	1.0
$\lvert\kappa_2\rvert$	1.0	2.0	1.0
$\lvert\kappa_1\rvert/\lvert\kappa_2\rvert$	1.0	2.0	1.0
λ_{1_x}	2.0	7.0	2.5
λ_{1_y}	2.0	7.0	2.5
λ_{1_z}	2.0	7.0	2.5
I	5.0	1.5	4.0
SSDO	4.0	1.5	4.0
$n \cdot v$	2.5	2.0	3.0
$\lvert\nabla(n \cdot v)\rvert$	1.5	2.0	3.0
$(\nabla(n \cdot v)) \cdot v$	1.5	2.0	2.0
$\lambda_1 \cdot v$	2.0	5.0	2.0
$\lambda_2 \cdot v$	2.0	5.0	2.0
z	2.3	1.0	4.0
x_i	2.3	1.0	4.0
y_i	1.5	1.0	4.0
n_{i_x}	2.0	4.0	2.5
n_{i_y}	2.0	4.0	2.5

Table 1: Feature weights. κ_1 and κ_2 are the first and second principal curvature directions, λ_1 and λ_2 are the first and second optimized principal curvature directions according to Hertzmann and Zorin [2000] (λ_{1_x} is the x-component of λ_1), I is the diffuse illumination (Lambertian shading), SSDO is the screen-space directional occlusion according to Ritschel et al. [2009], n is the surface normal, v is the viewing direction, ∇ is the gradient, z is the depth, x_i and y_i are the image-space coordinates, n_{i_x} and n_{i_y} are the x- and y-components of the image-space normal.

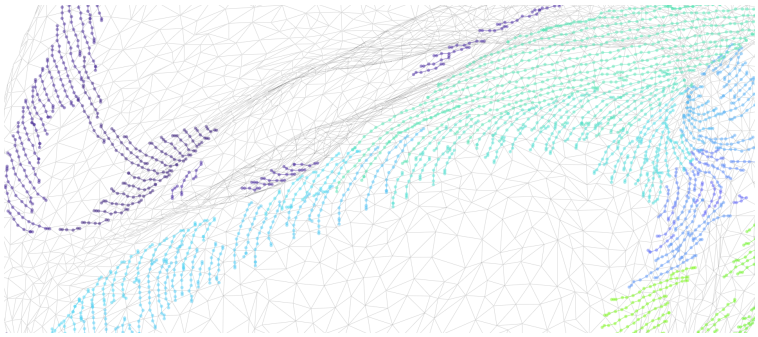
The described model of patch properties is prone to overfitting. Moreover, the sparse set of features makes it less accurate than the model of Kalogerakis et al. [2012]. We discuss the resulting limitations in more detail in Section 5.8. We proceed with describing our approach to learning the hatching stroke directions on the surface.

5.4.3 *Stroke Directions*

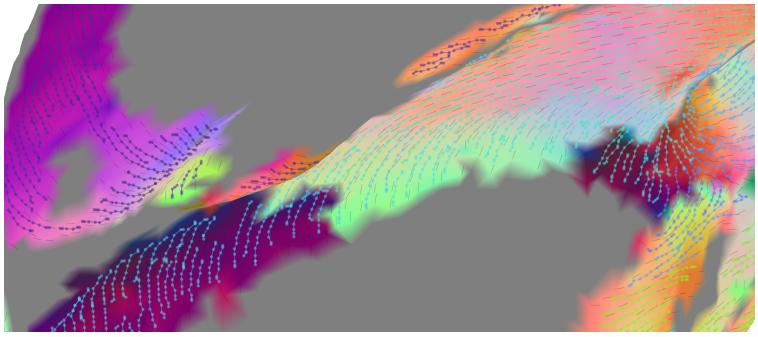
We learn and predict the locations of patches of example strokes on a global (mesh) level. We now descend one level in our style descriptor hierarchy and explain how we capture the directions of hatching strokes on a patch level. We do this by establishing a mapping of surface features to the directions of the example strokes reprojected onto the surface. In this way, we learn how the surface directions of the strokes drawn by the illustrator correspond with surface features. Applying this learned mapping in the synthesis stage yields an example-based patch-wise direction field (Section 5.5.1) which incorporates the directional characteristics of the learned hatching style.

We use the same set of features as described in Section 5.4.2 for learning the stroke directions, while weighting the directional features significantly stronger (see Table 1 for details). We use the optimized curvature direction field proposed by Hertzmann and Zorin [2000] as a reference direction field. We gain an object-space representation of the example strokes by reprojecting the detected example strokes (Fig. 5.7a) onto the the registered 3D model (Fig. 5.6b). An example for such reprojected strokes is shown in Fig. 5.9a. We then use regression analysis to learn a mapping from surface features to 3D stroke directions. We measure a scalar of the local stroke direction as the angle between the local stroke direction and the first optimized curvature direction in the tangent plane. With local direction we mean the direction of a segment of a stroke represented as 3D polyline. We train one regression function for each example stroke patch. For each vertex of a patch (see Fig. 5.8a) we gather and average the local stroke directions at the K nearest control points of the reprojected example strokes. This yields an example stroke direction field as shown in Fig. 5.9b. For the examples presented in this chapter, we used a value of $K = 5$. Eventually, the training data for learning the direction field function consist of one feature vector and one angle per vertex of a patch. We employ kernel ridge regression [Hoerl and Kennard, 1970] using radial basis function kernels for learning. We selected this learning method also by experiment, comparing it to radial basis function networks and relevance vector machines.

We complement the described learning of directional characteristics with a model of the distances between neighboring strokes.



(a)



(b)

Figure 5.9: Learning stroke directions. For each patch, we use (a) the reprojected stroke trajectories to create (b) an example stroke direction field. We then employ regression analysis to establish a mapping from surface features to example stroke directions.

5.4.4 *Stroke Distances*

While we perform the capture and reproduction of stroke directions on a patch level, we model the distances between neighboring strokes more locally on a stroke level. We here use the same reprojection setup as described in Section 5.4.3. For each example stroke, we learn the 2D distances from one of its neighboring strokes along its extent as a function of the surface features. In the synthesis, we use these distance functions to push strokes towards or away from their neighboring strokes during the tracing of stroke trajectories (Section 5.5.3). In this way we can recreate the learned patterns of local stroke interrelationship. This local modeling of stroke distance relationships results in stroke patterns that are less equidistant and less regular than the results of previous methods. According to the feedback of two professional medical illustrators, this controlled example-based irregularity enhances the aesthetic quality of the resulting hatching illustrations. This local modeling of stroke

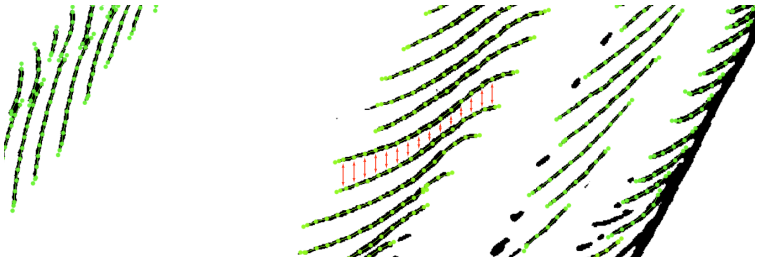


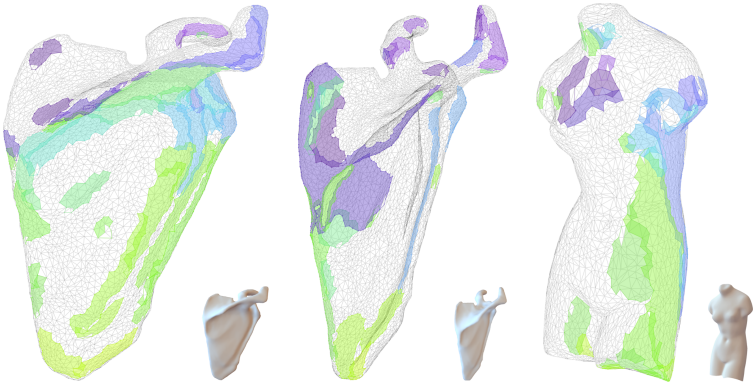
Figure 5.10: Learning stroke distances. For each detected stroke, we measure the 2D vertical distances (resp. horizontal distances for vertical strokes) to one of the neighbor strokes along the extent of the stroke. We then train a regression function to establish a mapping from surface features to the measured stroke distances.

distances is made possible by the detection of hatching strokes in the example that we described in Section 5.4.1.

For learning the stroke distance functions, we again use the same set of features as explained in Section 5.4.2 and put most emphasis on the image-space coordinates and on the diffuse and ambient lighting (see Table 1). For a horizontal example stroke, we measure the 2D vertical distance to its lower neighbor stroke at every control point. This distance measurement is illustrated in Fig. 5.10. We interpolate the surface feature vectors measured at the vertices to gain interpolated feature vectors at the reprojected control points. We use barycentric coordinates for a component-wise interpolation of the feature vectors. Using this data, we train a regression function for each stroke. We here also employ kernel ridge regression with radial basis function kernels.

5.4.5 *Summary*

After running the described learning procedures, we have a description of the drawing style stored in the following way. The coordinates and widths of the example strokes are stored in a text file, grouped in patches. We make use of the dlib library [King, 2009] for performing the described machine learning methods. The patch classifier, the direction field functions, as well as the stroke distance functions are stored as dlib decision functions. The example stroke data and learned functions can thus be loaded from disk and used in the synthesis stage of our method.



(a) (b) (c)

Figure 5.11: Adaptive surface patches. We generate a dynamic example-based mesh segmentation and grow adaptive surface patches from the resulting labeling. The figure shows the patches generated for (a) the input mesh and view, (b) a different view, and (c) for a different mesh. We use these surface patches to generate hatching strokes. The surface patches can also be edited directly via brushing interactions (Section 5.6).

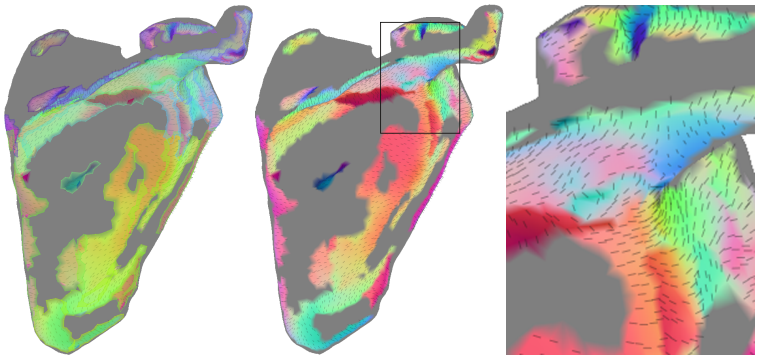
5.5 HATCHING SYNTHESIS

In this section we detail how we apply the learned model to a target 3D mesh in order to synthesize a hatching illustration by example.

5.5.1 *Adaptive Patches*

The first step in our synthesis pipeline is to apply the patch classifier described in Section 5.4.2 to predict a patch label at each vertex of the target mesh. At each vertex, we input the surface feature vector into the patch classifier, which outputs a patch label for the current vertex. This classification procedure yields a real-time dynamic segmentation of the target mesh into surface patches which incorporate the recorded global properties of the hatching style that we aim to reproduce. It is dynamic because the segmentation is regenerated for every frame, and gives a new result according to the new view direction and lighting conditions.

Based on this vertex labeling, we grow adaptive patches on the surface in order to gain an explicit geometric representation of the surface patches. We let surface snakes evolve on the mesh as proposed by Bischoff et al. [2005] to gain the patches. We employ surface snakes following the predicted patch labels to collect the connected vertices and faces of every patch label, and to establish an explicit representation of the boundary of each patch. Our surface snakes do not evolve iteratively and do not move according to a velocity, in contrast to the snakes described by Bischoff



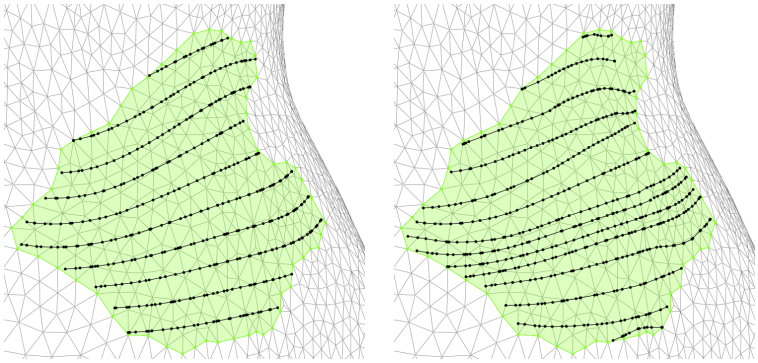
(a) (b) (c)

Figure 5.12: Example-based direction field. We apply the learned direction field functions to infer an example-based stroke direction field. Each surface patch is associated with a different direction field function and the inference is performed patch-wise. (a) shows the surface patches overlayed over the (b) inferred direction field, (c) shows a detail of (b). These directions are the basis for tracing stroke trajectories on the surface.

et al. [2005]. Our surface snakes evolve recursively and move the full length of an edge per recursion. Our snake control points split at each vertex and stop their evolution at vertices that are assigned a patch label that is different from the patch label gathered by the current snake. We prevent the generation of too small patches by omitting all patches that are formed by a surface snake whose number of control points is below a threshold. Fig. 5.11 shows the resulting adaptive surface patches for the input mesh, for a different view, and for a different mesh. We denote them as adaptive because they adjust to the current viewing and lighting conditions. When the object or the light sources are transformed, the patches move along the surface. These adaptive patches embedded on the surface are the basis for the following steps in our hatching synthesis pipeline. The explicit representation of hatching regions as surface patches also makes it possible to realize brushing interactions that allow users to directly adjust the resulting hatching illustration (see Section 5.6).

5.5.2 *Example-based Direction Field*

For each adaptive patch, we apply a direction field function as described in Section 5.4.3 to infer an example-based stroke direction field. Every adaptive patch is associated with an example stroke patch and uses its direction field function. Thus, a different direction field is inferred for each adaptive patch. The inferred direction field incorporates the directional characteristics learned from the example illustration. The inference takes place at the vertices of the target mesh. One angle is inferred for each



(a) (b)

Figure 5.13: Stroke distance control. The interrelationship of (a) strokes following the direction field is (b) enhanced by applying the learned distances functions.

vertex. The angle is obtained by evaluating the direction field function using the respective feature vector as argument. We rotate the optimized curvature direction by the inferred angle on the tangent plane to obtain the final stroke direction. We use the resulting example-based direction field for tracing the trajectories of hatching strokes on the surface. This patch-wise direction field inference is similar to the segment-wise direction inference in image space that was presented by Kalogerakis et al. [2012]. Our object-space representation has the advantage, however, that it enables us to provide object-space brushing interactions for editing the reference direction field (see Section 5.6). This reference field retouching allows users of our system to flexibly and directly adjust the trajectories of hatching strokes on the surface.

5.5.3 *Stroke Tracing and Distances*

We trace stroke trajectories on the surface by integrating the inferred stroke directions. At the same time, we use the learned distance functions as explained in Section 5.4.3 to control the distances between strokes in image space. In this way we gain strokes which follow the example-based direction field and which recreate the learned patterns of stroke interrelationship. This means that the generated stroke trajectories deviate from each other and approach each other in similar ways as the strokes in the example, introducing controlled example-based distance irregularities. Each stroke is associated with an individual distance function. Our local approach to transferring stroke distances results in a more detailed transfer of stroke interrelationships as compared to the global pixel-based approach of Kalogerakis et al. [2012]. This results in hatching strokes that

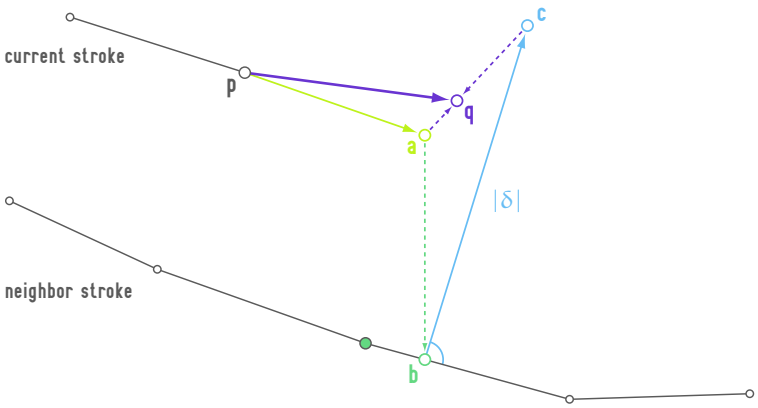


Figure 5.14: Stroke distance control in 2D. Starting from the current control point p, the next control point q is determined based on a direction-based proposal a and a distance-based proposal c. First, the direction-based proposal a is gained by integrating the surface direction field by one step and projecting the point to 2D. Next, the vertical intersection b with the neighbor stroke is found by intersecting the neighbor stroke with a vertical line through a. A distance δ is predicted by evaluating the distance function of the neighbor stroke. This prediction requires the lookup of 3D surface properties, which is achieved by using the surface location that corresponds to the neighbor stroke control point closest to b. The distance-based proposal c is then found at the predicted distance δ along a perpendicular on the local direction of the neighbor stroke starting from b. Finally, the next stroke control point q is calculated by linearly interpolating a and c. The interpolation can be controlled with one user-tunable parameter. For creating the result images in this chapter, we used a value of 0.3, meaning that each new position is calculated to 30 percent by distance from the neighbor and to 70 percent by direction.

are less regular and less equidistant, which enhances the hand-drawn character of the hatching illustrations generated with our approach.

The stroke control points are embedded on the surface, living within triangles of the mesh and on mesh edges. During the tracing within a triangle, we interpolate the stroke directions inferred at the triangle's vertices using barycentric coordinates. A trajectory is stopped when it reaches the boundary of an adaptive patch. During the tracing in object space, we control the stroke distances in image space, in contrast to previous object-space techniques [Zander et al., 2004]. While tracing a stroke, we repeatedly evaluate its associated distance function. We use the predicted distances from the neighboring stroke as a second component influencing the position of each new control point, complementing the directions from the stroke direction field. The effect of applying these stroke distance functions is demonstrated in Fig. 5.13. The distance-controlled tracing algorithm is illustrated in Fig. 5.14.

We place strokes incrementally, according to a seeding strategy adopted from Jobard and Lefer's [1997] streamline algorithm. We chose the algorithm of Jobard and Lefer [1997] as a basis for our stroke generation due

to the following reasons. First, the algorithm facilitates the generation of equidistant lines within a direction field. Second, the algorithm permits to derive new strokes from existing ones. Third, Jobard and Lefer's [1997] streamline algorithm implements a distance control mechanism via a spatial search structure. We use the distances predicted by the learned stroke distance functions for seeding new strokes from existing ones as well as for terminating strokes which come too close to each other. The strokes are rendered as follows.

5.5.4 *Stroke Rendering*

We create 2D textured triangle strips from the 3D stroke trajectories for rendering the hatching strokes. This involves two attempts of transferring low-level stroke properties. With those attempts we try to reproduce the width and shape of the pen-and-ink strokes in the example image.

First, we use the stroke widths measured along the detected example strokes for creating the 2D stroke geometry. Our stroke detection allows us to measure the stroke widths along the extent of each detected stroke and to parameterize the measured stroke widths depending on the position on the trajectory. We use these parameterized stroke widths for creating triangle strips of varying width during stroke rendering. At each control point of a stroke to be rendered, we calculate the current width using the measured and parameterized stroke width. Examples for such textured triangle strips are shown in Fig. 5.15b.

As a second attempt of transferring low-level properties, we texture the resulting strokes using the entire example image as texture. This allows us to render strokes that simulate 'real' pen-and-ink drawing marks. We realize this texture mapping by creating texture coordinates that tightly enclose individual strokes in the example image. To gain the required texture coordinates, we first map the detected stroke trajectories and measured stroke widths to texture space using a simple linear mapping. We then generate texture coordinates for each detected example stroke by fitting a spline through the stroke control points and generating pairs of texture coordinates at perpendicular offsets from the spline. The offsets are calculated using the measured stroke widths. In other words, we map the detected stroke trajectories to texture space and inflate them according to the measured stroke widths. The resulting texture coordinates tightly enclose the detected example strokes, as shown in Fig. 5.15a. The image in Fig. 5.15c shows a detail section of Fig. 5.15a with a color-coded rendering of the texture coordinates. The u-coordinate is mapped to green color and the v-coordinate to blue color. The colors are normalized to the depicted section of the texture space for readability.

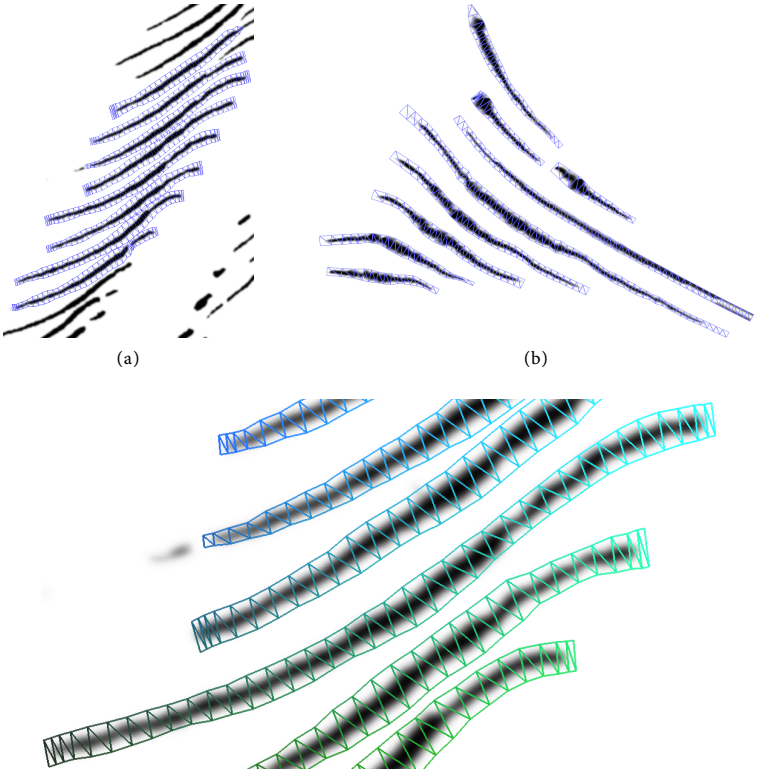


Figure 5.15: Example-based stroke widths and stroke texturing. (a) shows a section of the texture space (the texture image with superimposed texture coordinates) and (b) shows a set of rendered strokes in image space. We use the entire example image as texture and create (a) texture coordinates that tightly enclose strokes in the example image. This is done by mapping the trajectories of the detected strokes to texture space and inflating them according to the measured stroke widths. Each rendered stroke in (b) is assigned one texture-space stroke from (a). The measured stroke widths are used for creating both the geometry of the (a) texture-space strokes as well as of the (b) rendered strokes. (c) shows a detail section of (a) with a color-coded rendering of the texture coordinates. The u-coordinate is mapped to green color and the v-coordinate to blue color. The colors are normalized to the depicted section of the texture space for readability.

We perform antialiasing for on-screen display via blurring and mipmapping the example texture and via supersampling. We make use of texture interpolation for supersampling. The supersampling is achieved via first rendering the result image to a texture that is four times the size of the viewport. The result is subsequently rendered to the viewport as a textured quad. We threshold and binarize the images for the result images in this document, which also helps to reduce texturing artifacts.

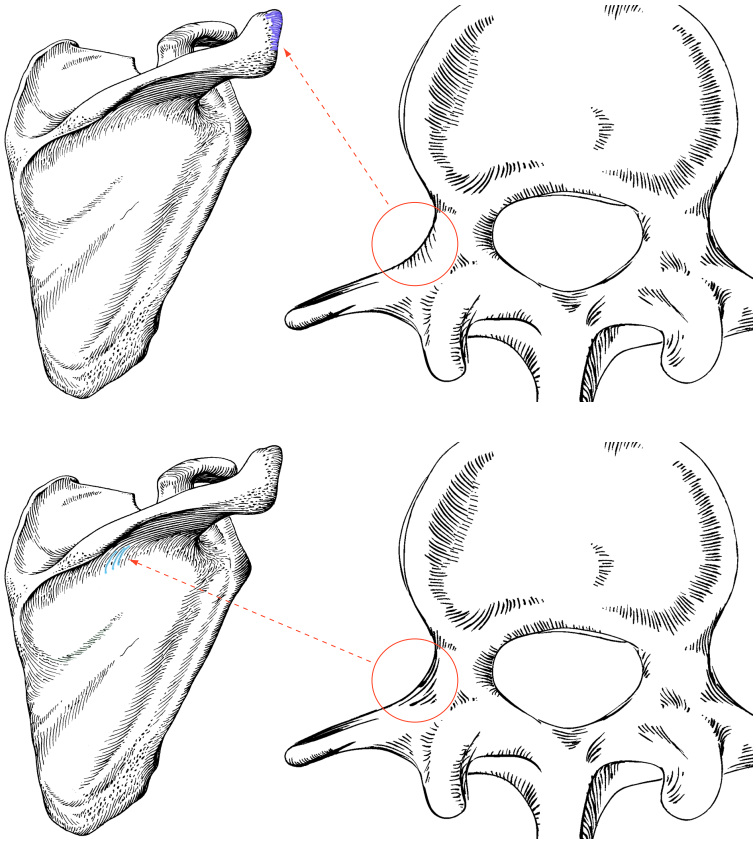


Figure 5.16: Mapping interaction. Each patch of hatching strokes in the result illustration (right) can be assigned a different patch of strokes in the example illustration (left).

5.6 INTERACTION WITH THE HATCHING ILLUSTRATION

The processes described in the previous section automatically synthesize a hatching illustration of a 3D model. To complement this automatic generation, we allow users to interact with the illustration in four different ways. Users can modify which type of strokes are generated within a surface patch, adjust the hatching angle, retouch the reference direction field with brushing tools, and brush with patches of hatching strokes. This set of interactions allows users to adjust the illustrations according to their requirements and aesthetic judgment and, thus, to enhance the aesthetic appearance of the resulting illustrations.

First among the interactions, users can modify which type of strokes are created within a particular region. We realize this modification by assigning a different example stroke patch to a surface patch. The reassignment leads to the usage of a different direction field function, different

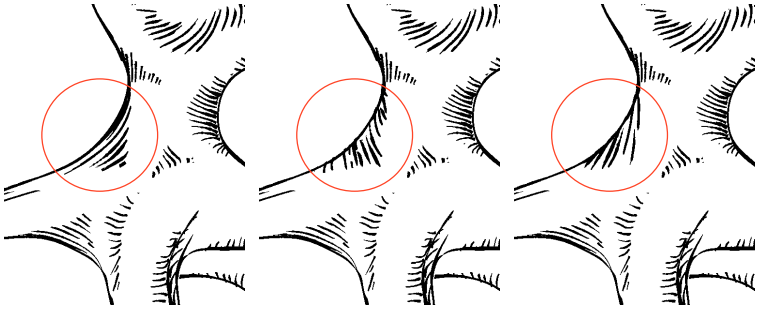


Figure 5.17: Hatching rotation interaction. Users can control the general hatching direction of each patch of hatching strokes. The strokes in the highlighted patch are rotated.

stroke distance functions as well as different stroke widths and textures. This interaction, thus, results in a different appearance of hatching strokes within an adaptive patch. Fig. 5.16 shows the effect of such an interaction.

Second, users can control the hatching angle of each individual patch of hatching strokes. We realize this interaction by adding a user-controlled angle to the stroke direction angle that is inferred during the direction field inference. This interaction allows users to adjust the general hatching direction of all strokes within one patch, while the strokes still incorporate the learned and reproduced directional characteristics (see Fig. 5.17).

Third, users can modify the stroke trajectories by retouching the reference direction field (the optimized curvature direction field) with brushing interactions. Fig. 5.18 shows an interaction sequence for this object-space direction field retouching. Modifying the reference direction field results in a modified inferred direction field and, thus, in modified stroke trajectories. We provide three different direction field editing tools. These three radial brushing tools operate with a user-controllable brush size, strength, and hardness (a Gaussian attenuation of modification intensity dependent on the distance from the brush center). The first of the three tools is a clone stamp tool that transfers the reference directions from a source area to the brushing area (Fig. 5.18a–Fig. 5.18c). Equally to the clone stamp in Adobe Photoshop®, the source area is selected initially and moves relatively to the cursor location. This clone stamp tool allows users to conveniently retouch singularities and discontinuities of the reference direction field and to transfer reference directions from one surface location to another. As a second operator, a blur tool averages the reference directions in the brushing area and allows users to smooth stroke trajectories (Fig. 5.18d–Fig. 5.18f). Finally, a rotate tool rotates the reference directions by a user-defined angle on the tangent plane (Fig. 5.18g–Fig. 5.18i). This tool allows users to freely adjust the direction of stroke trajectories. Together, the described brushing interactions on

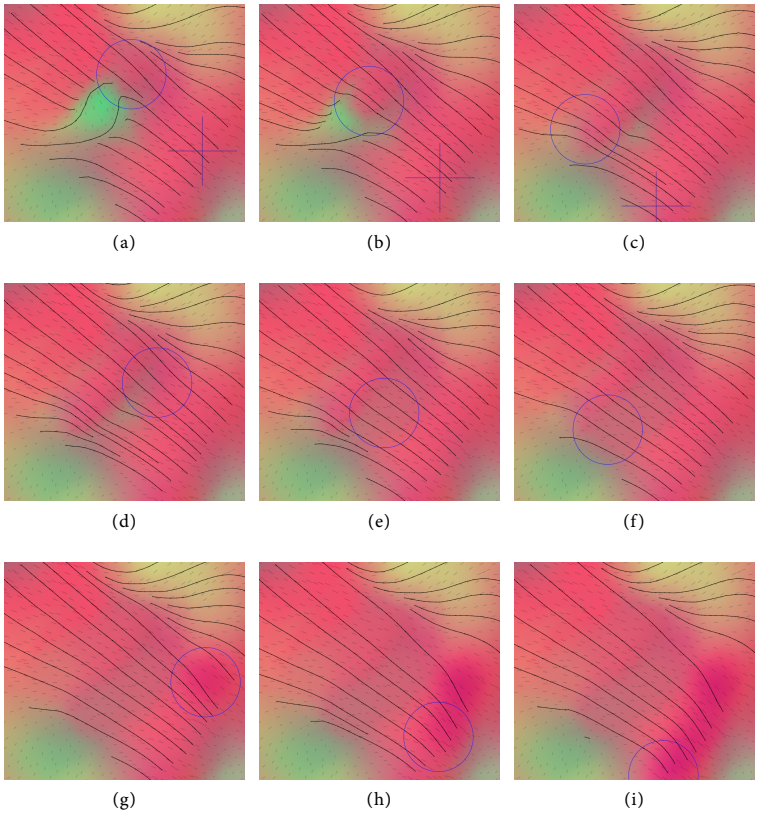


Figure 5.18: Direction field retouching. The sequence starts with (a)–(c) the clone stamp tool. The user retouches the reference direction field to adjust the stroke trajectories in the central green area. The clone stamp transfers the directions from the source location (indicated by the crosshairs) to the brushing location. Next, the user employs (d)–(f) the blur tool to smooth the discontinuities that were introduced with the clone stamp (the discontinuities are visible as the remainder of the green spot in the center that causes the bumps in the stroke trajectories). Finally, the user employs (g)–(i) the rotate tool to bend the tips of the stroke trajectories. The colors show the reference directions mapped to RGB. The needles indicate the reference direction at each vertex. The stroke trajectories shown in black are updated on the fly.

the reference direction field permit users to freely adjust the stroke trajectories to their needs. The tools can be used for coarse adjustments, such as modifying the general hatching direction of an entire patch, or fine-grained modifications, such as bending the tip of an individual stroke. This flexible control over object-space stroke trajectories is a novelty of our approach that improves upon the static nature and the lack of control over the result of existing hatching methods.

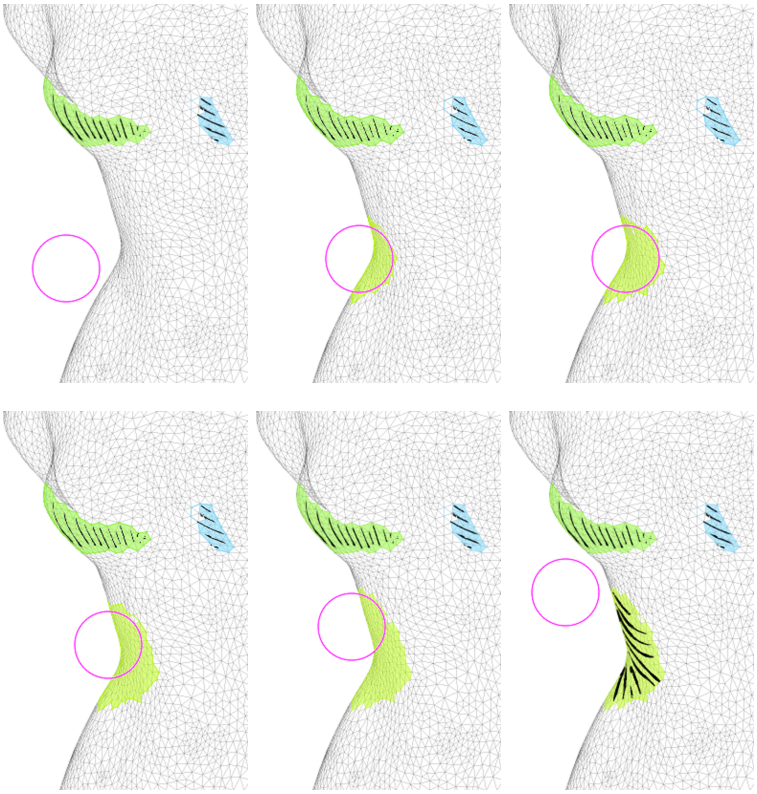


Figure 5.19: Brushing interaction. The user brushes a patch of hatching strokes onto the surface. The strokes for the brushed patch are generated after the interaction. This interaction also allows users to reshape or remove hatching patches.

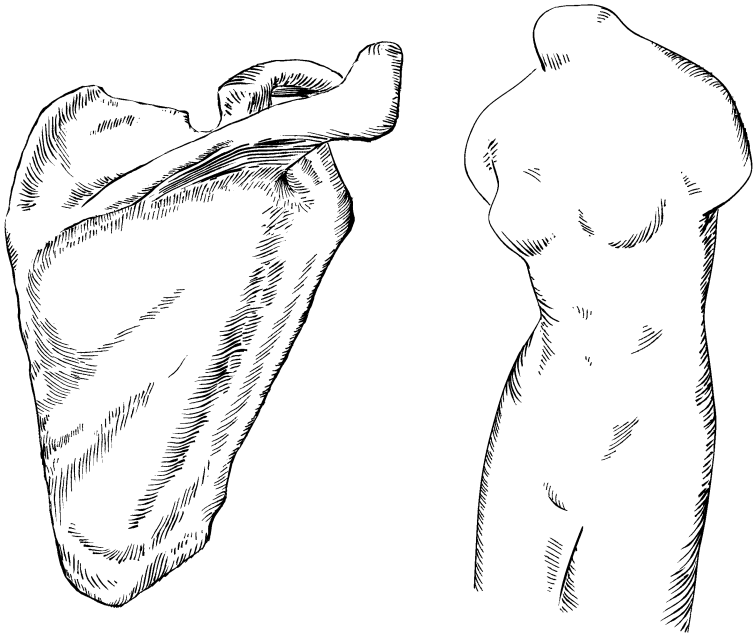
Fourth, users can brush with patches of hatching strokes onto the surface. Fig. 5.19 shows an image sequence for this interaction. The brushing interaction is realized by interactively altering the automatic mesh segmentation (see Section 5.5.1) via brushing. This brushing interaction is implemented as the assignment of a particular patch label to mesh vertices within the brushing area. Changing the mesh segmentation in this way effectively results in adding, modifying, or removing adaptive patches. The hatching strokes within the modified patches are re-generated on the fly. This gives users the possibility to interactively modify the hatching illustration to achieve the desired result. In some cases, the automatically generated hatching patches are suboptimal because they cover unwanted areas and are not existent in other areas where strokes are desired. Users can then adjust the illustration with the described brushing interaction to achieve aesthetically more pleasing and more effective results. Users can as well disable the automatic prediction of hatching regions and start

from scratch to freely brush hatching patches onto the surface according to their requirements. The stroke directions and distances, however, are always inferred automatically.

Apart from the mapping and brushing interactions described above, there are numerous parameters in our system to tune the result. These involve simple parameters which could be exposed to novice users, such as stroke width and stroke distances. Other parameters allow expert users of our system to adjust the result in more detail, such as the feature weights that are used within the three different machine learning components.

The described interactions override the automatic prediction of hatching properties. On the one hand, these interactions serve for dealing with limitations of our automatic style transfer mechanisms (see Section 5.8). On the other hand, the interactions effectively combine the advantages of automatic hatching and human creativity. Our interactions, therefore, represent novel tools for the semi-automatic generation of hatching illustrations in the spirit of existing semi-automatic methods for non-photorealistic rendering [Winkenbach and Salesin, 1994; Salisbury et al., 1994; Deussen et al., 2000]. The interactions provide users with a means to directly and intuitively specify or modify the regions where strokes are placed, which type of strokes are generated in which region and at which direction on the surface. These interaction capabilities make an employment of our method in a creative environment more likely than the usage of a fully automatic and static method. Furthermore, the proposed integration of example-based hatching with interaction capabilities permits users without a background in hatching to create illustrations they would otherwise not be able to create.

As discussed in Chapter 2 and in Chapter 4, the strategy of combining machine learning and interactivity for implementing control over the result does allow us to let the result of our hatching method be influenced by human virtuosity in a twofold way: by the machine as well as by the user. This strategy improves upon the aesthetic quality of the images that are achievable with our approach.



(a) (b)

Figure 5.20: Illustrations using the hatching style learned from the shoulder blade illustration in Fig. 5.6a. The depicted objects are (a) the shoulder blade mesh we use for learning and (b) the venus mesh. Both illustrations are created semi-automatically by applying the interaction methods described in Section 5.6. Contours are curvature-controlled image-space contours as proposed by Bruckner and Gröller [2007], but any other silhouette technique [Isenberg et al., 2003; Bénard et al., 2012] could be chosen.

5.7 RESULTS AND DISCUSSION

In this section we show and discuss some results generated with our method. We first present some results of applying the drawing style learned from the shoulder blade illustration shown in Fig. 5.6a. We then show the input and results of transferring the hatching style of a second example illustration. All the result images are created semi-automatically making use of the user interactions described in Section 5.6. For example, manual adjustments include the mapping of example stroke patches, the modification of the general hatching directions of individual patches, the retouching of the reference direction field to refine stroke trajectories, and the adjustment of the hatching areas via brushing. These manual adjustments permit us to enhance the visual appearance of the results by considering illustration and aesthetical issues during the image synthesis. These are issues such as lighting, shape perception and shape communication, and the interplay of different stroke groups.

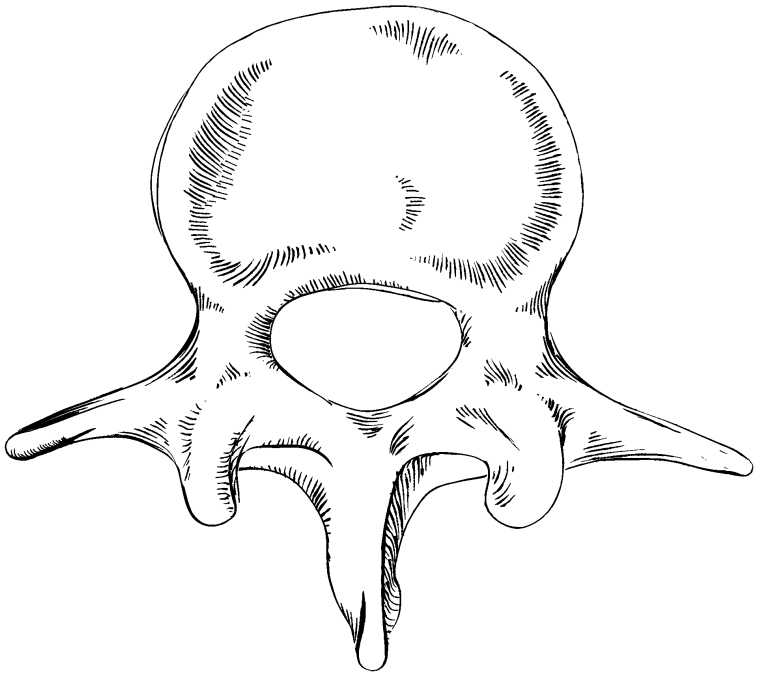


Figure 5.21: Illustration of the third lumbar vertebra using the hatching style learned from the shoulder blade illustration in Fig. 5.6a.

Fig. 5.20–Fig. 5.22 show results of transferring the hatching style learned from the shoulder blade illustration in Fig. 5.6a. While creating Fig. 5.20a, we aimed to match the example illustration (Fig. 5.6a). We also transfer the learned hatching style to new objects. The venus statue illustration in Fig. 5.20b, the vertebra illustration Fig. 5.21, and the hip bone illustration in Fig. 5.22 demonstrate that we can transfer the learned hatching style to different target meshes.

For comparison, we also apply our method to the pen-and-ink hatching style of a different illustrator. Fig. 5.23a shows this second example illustration. It is a hand-drawn pen-and-ink illustration of the trap of a carnivorous tropical pitcher plant which was created for a previous study of Isenberg et al. [2006]. Together with the segmentation image shown in Fig. 5.23c and the registered 3D model in Fig. 5.23b, we use the pitcher plant illustration as input for learning a second hatching style. The preparation of the learning input here happened slightly differently. In contrast to the shoulder blade illustration (Fig. 5.6d) which we scanned in from an anatomy textbook [Dauber et al., 2005], the pitcher plant illustration was drawn with the illustrator using a rendering of a 3D pitcher plant model

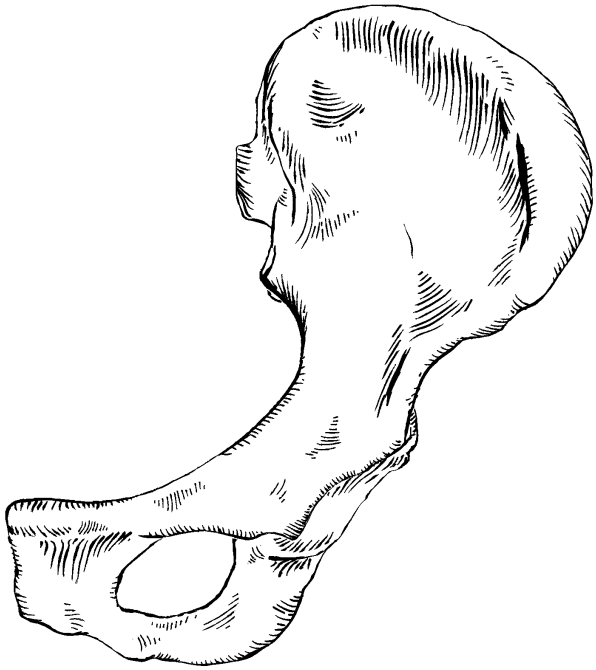


Figure 5.22: Illustration of a hip bone using the hatching style learned from the shoulder blade illustration in Fig. 5.6a.

as a master. In the study of Isenberg et al. [2006], illustrators were shown renderings of 3D models and created illustrations using these renderings as masters. In this case the 3D model was, thus, already close to the drawn image and had to be only slightly adjusted. The shoulder blade 3D model, in contrast, had to be sculpted from a random shoulder blade model that did not match the example illustration as closely. Fig. 5.24–Fig. 5.27 show results of transferring the hatching style learned from the pitcher plant illustration in Fig. 5.23a to various objects.

The results show that many characteristics of the example illustration styles are reproduced. For example, patterns of hatching strokes within the patches of strokes are reproduced. These patterns emerge from both the directions and the distances of strokes. Regarding the directions, patterns emerge from the quasi-parallel trajectories of the strokes and the way individual strokes deviate from these common directions. Patterns with respect to the distances emerge from the sequences of distances between the strokes. This applies to both the sequences of distances between strokes within a patch and the sequences of distances along the extents of neighboring strokes. The latter transfer patterns of the rela-

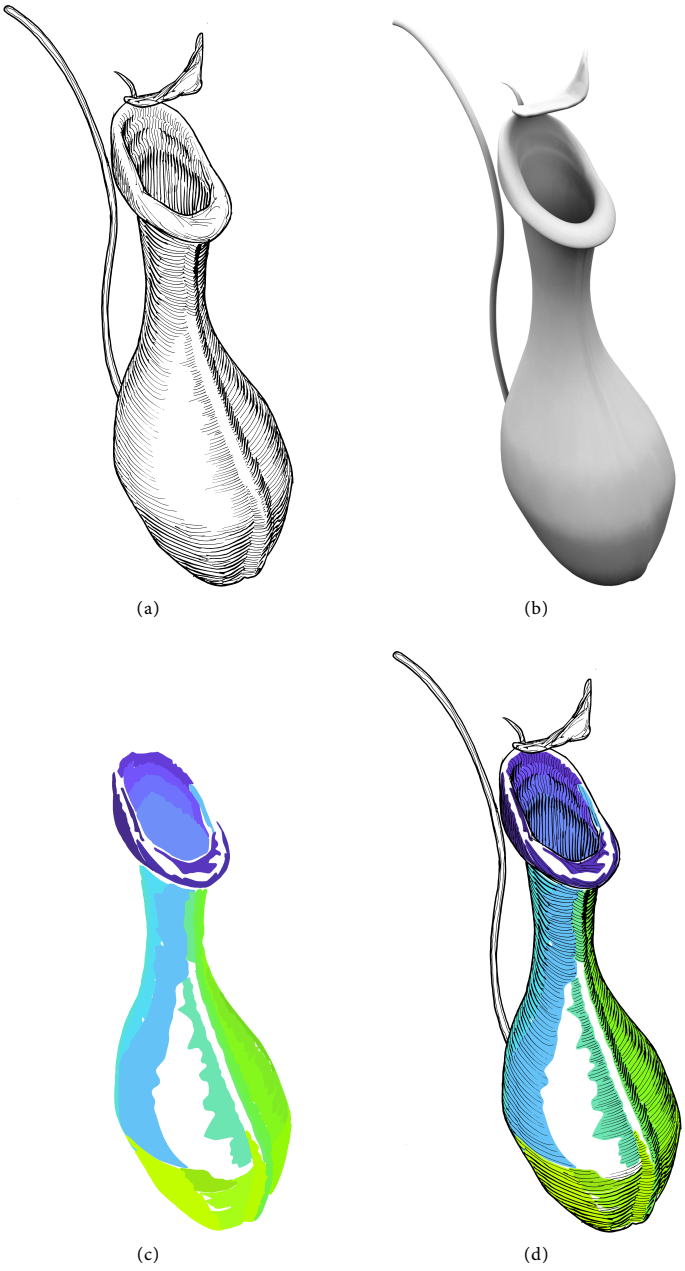
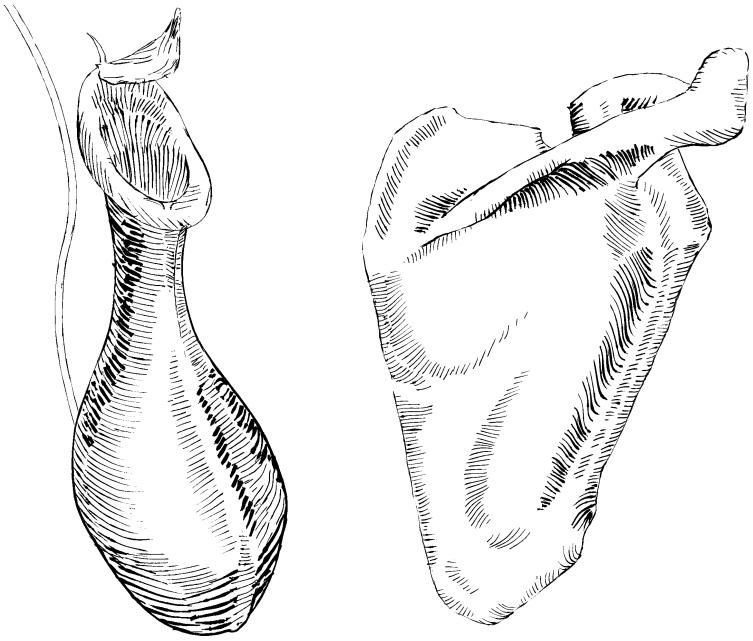


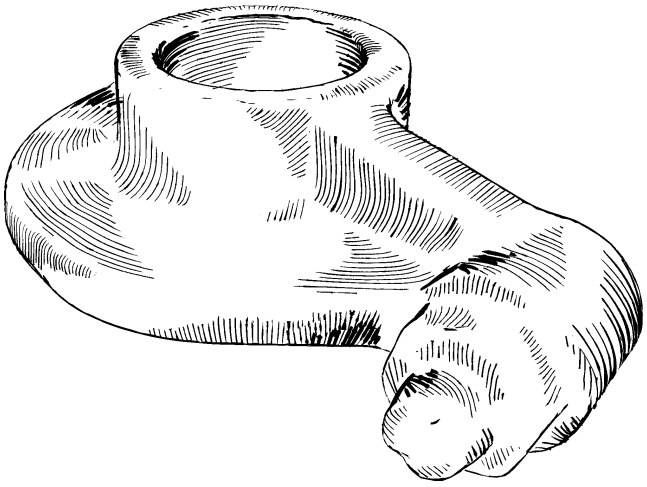
Figure 5.23: (a) A second example illustration from a study of Isenberg et al. [2006]. It shows the trap of a tropical pitcher plant. (b) shows the registered 3D model, (c) the segmentation image and (d) an overlay of the example with the segmentation image.



(a) (b)

Figure 5.24: Illustrations using the hatching style learned from the pitcher plant illustration shown in Fig. 5.23a. The depicted objects are (a) a pitcher plant (intended to match Fig. 5.23a) and (b) a shoulder blade.

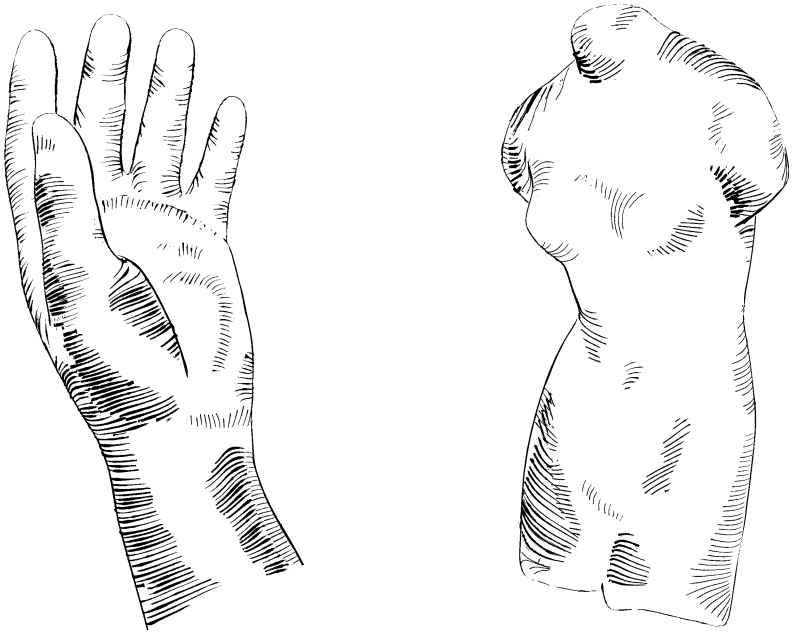
tionship between neighboring strokes, meaning the way in which neighboring strokes approach and deviate from each other. This controlled example-based stroke irregularity recreates some of the visual appearance of the example illustrations and has a considerable positive effect on the 'character' of our results. Furthermore, the directions of the resulting hatching strokes on the surface appear to be similar to the strokes' surface directions in the example image. The example-based direction field inference does reproduce the way in which strokes are following the surface. Using this direction field to trace strokes on the surface thus results in hatching strokes that visually model the depicted surface in similar ways as the strokes in the example illustration. Furthermore, the interplay of groups of different stroke types helps reproducing the visual appearance of the example illustrations. This effect is supported by our stroke rendering approach, which simulates real pen-and-ink marks to a certain extent. However, not all of the characteristics can be faithfully reproduced. The overall appearance of our results still shows differences between the originals and the synthesized illustrations. We elaborate on these shortcomings in Section 5.8.



(a)

Figure 5.25: An illustration of the rocker arm mesh using the style learned from the pitcher plant illustration in Fig. 5.23a.

When comparing our results to the results presented by Kalogerakis et al. [2012] (see Fig. 5.3), we make the following observation. Three of their example styles use quite uniform and regular styles (Fig. 7–9 in [Kalogerakis et al., 2012]), while two example styles use more irregular styles (Fig. 6 and 10 in [Kalogerakis et al., 2012]). We observe that the irregular hatching styles are not transferred as faithfully as the uniform styles. In particular local characteristics of the irregular styles, such as irregularities of the stroke trajectories or the relationship of neighboring strokes, cannot be reproduced that well and are lost in the transfer process. The uniform hatching styles, however, are reproduced very accurately. The overall appearance of the synthesized hatchings visually match the example illustrations impressively well. We do not achieve the same transfer accuracy with our automatic method. The described observation suggests that, in general, uniform and regular drawing styles can more easily be reproduced than irregular and complex styles. We now observe that the drawing styles that Kalogerakis et al. [2012] employ are, in comparison, much more uniform and regular than the drawing styles we try to reproduce. The strokes in their example drawings have a strong correlation with the surface curvature, and are mostly equidistant and parallel. The strokes in our example illustrations are much more complex and irregular, which makes it all the more difficult to faithfully transfer the drawing styles. Although we might not reach the same transfer accuracy as Kalogerakis et al. [2012], we can transfer some characteristics of the complex illustration style. In particular, local characteristics can

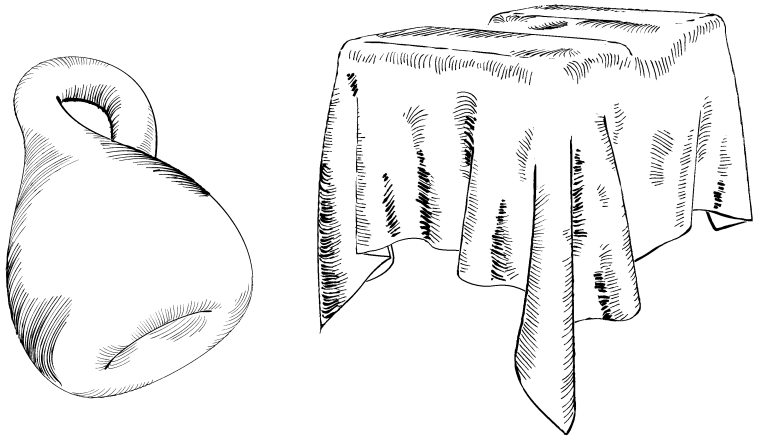


(a) (b)

Figure 5.26: More illustrations using the hatching style learned from the pitcher plant illustration in Fig. 5.23a. The depicted objects are (a) a hand and (b) a venus statue.

be transferred more faithfully with our style transfer model, because our model explicitly takes them into account. As a result, our approach facilitates the transfer of more stylistic detail, such as the distance relationship between neighboring strokes. Furthermore, the explicit handling of groups of strokes allows us to transfer drawing characteristics that are embodied in patches of strokes and in the relationships of these patches.

Apart from that, Kalogerakis et al. [2012] use a vast set of geometric features, which is another reason that their style transfer model can more accurately reproduce the overall appearance of the example styles. Using such many features, however, severely affects the performance of their algorithm. Kalogerakis et al. [2012] name 5 to 10 hours learning time and 30 to 60 minutes synthesis time on an Intel Core i7 processor. Our method takes 1 to 2 minutes for learning and 0.4 to 30 seconds for synthesis on an Intel Core 2 Duo processor. As a pixel-based approach, the performance of their method depends on the resolution of the result image, while the performance of our method is virtually independent of the output resolution. With 30 seconds per frame in the worst case, our method still does not reach real-time frame rates, but we managed to keep it real-time applicable for hardware generations of the near future.



(a) (b)

Figure 5.27: More illustrations using the hatching style learned from the pitcher plant illustration in Fig. 5.23a. The objects are (a) a Klein bottle and (b) a two box cloth.

Another advantage of our approach over that by Kalogerakis et al. [2012] is the possibility for interaction. Our interactions permit users to adjust the illustration in order to achieve more aesthetically pleasing results. Such adjustments are used, e. g., to reshape large continuous hatching regions, to add strokes to blank regions, or to make use of the aesthetics of combining different patches of strokes. The fully automatic approach of Kalogerakis et al. [2012] does not facilitate such adjustments. Our method, in contrast, can serve as the basis for a tool for artists and illustrators due to its interaction capabilities.

Together, the differences between our approach and that of Kalogerakis et al. [2012] can be summarized as the following: Kalogerakis et al. [2012] present a fully automatic approach with a high style transfer accuracy for uniform hatching styles. We, in contrast, present a semi-automatic approach with a lower transfer accuracy for complex hatching styles which has the capability to enhance the results via user interactions.

We make a step towards the computer generation of pen-and-ink hatching illustrations that incorporate human virtuosity and illustration skills. Even if our approach cannot capture and reproduce all the stylistic properties of the example illustrations, it does reproduce many of their characteristics. The interaction capabilities of our approach facilitate a further enhancement of the hand-drawn appearance and allow the results to be influenced by human creativity and virtuosity. For these reasons, we gain result images that look less synthetic and less uniform, and arguably exhibit more 'character' than the results of previous methods, which are either not example-based or not interactive.

Our patch-based approach to hatching can also easily be extended to achieve crosshatching. We realize this by adding another layer of hatching patches which use stroke directions at a user-controllable angle to the inferred stroke directions. Although our target illustration style does not use crosshatching, the illustrations resulting from this extension still have a certain aesthetic appeal. Fig. 5.28 shows an example of a crosshatching illustration achieved in this way.

We showed our results to two professional medical illustrators to gain informal user feedback. Both illustrators were impressed by the aesthetic quality of our illustrations. One of the illustrators described our results to have a *'lively appearance and not stiff like a digital feeling.'* The other illustrator commented positively that our *'method can produce good 'pen&ink' illustrations in far less time than a hand drawn illustration.'* One of the illustrators informed us that she would be interested in working with a system such as ours: *'I definitely could imagine to work with a method like yours! I would really appreciate a tool like that.'* This illustrator also stated that she lacked functionality comparable to our method in the software she is using. Furthermore, both illustrators stated that the manual creation of pen-and-ink hatchings similar to ours is so tedious and time-consuming that the required time prohibits them to create such illustrations for customers. The illustrators appreciated the possibility to rapidly create hatching illustrations with the computer while still having interactive control over the result: *'I totally agree that the interactive control over the result is beneficial for illustration purposes.'*

5.8 LIMITATIONS

The results presented in the previous section show that we can capture and reproduce characteristics learned from a hand-drawn illustration. In particular, we can faithfully reproduce local characteristics such as the distance relationship between neighboring strokes along their extent. In combination with the interaction methods, this style transfer offers new possibilities for creating pen-and-ink hatching renderings. The results also show, however, that our style transfer model suffers from certain accuracy issues. We believe that the automatic reproduction of human virtuosity is always limited. The project described in this chapter is an example for the insight that the general limitation of reproducing human virtuosity can be circumvented by combining automatic style transfer methods with user interactions. One reason that our learning methods do not fully capture the example style is that we use a limited number of surface features, as discussed in the previous section. This could be improved upon by including more features, although this would come with the price of reduced performance. It is evident from the work of

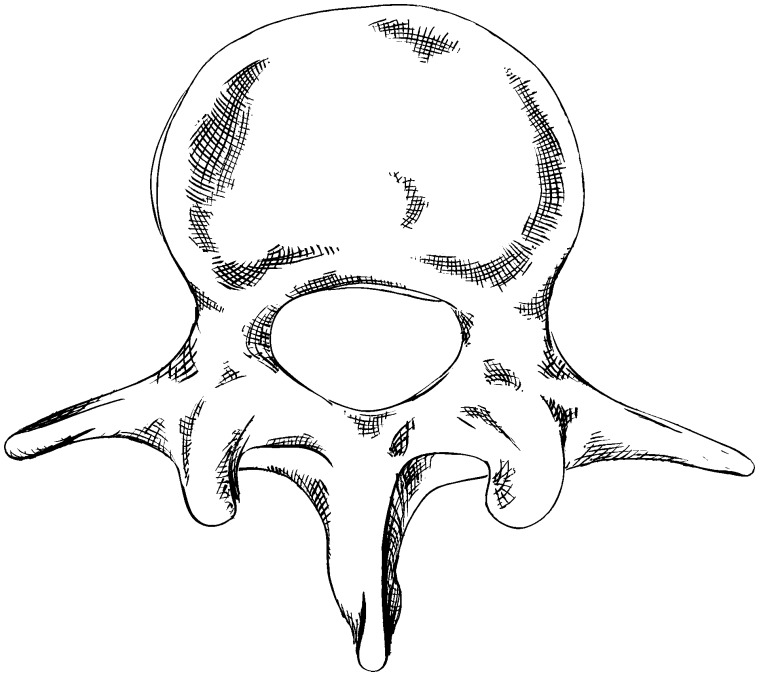


Figure 5.28: Crosshatching illustration of the third lumbar vertebra using the hatching style learned from the shoulder blade illustration in Fig. 5.6a.

Kalogerakis et al. [2012] that an extensive set of surface features yields more robust style transfer results but is slower. The performance gained by using a small set of features, however, allows us to realize a semi-automatic illustration system with responsive interactions.

Another problem is that our learning approach suffers from overfitting to one specific setup. We use just a single example illustration for learning a hatching style. All our drawing style descriptors are thus conditioned to one specific setup of viewing, lighting, and geometry which holds for this single illustration. This overfitting results in the problem that the hatching properties we infer for viewing and lighting situations other than the training setup as well as for other shapes do not match the hatching properties found in the example drawing. The described overfitting problem has most negative impact on the globally learned patch properties, which is apparent in Fig. 5.11. The surface patches inferred for the training setup match the regions of the segmentation image very well. For other viewing and lighting situations, however, the patches do not always match the regions which we assume the creator of the example illustration would have used. This is one of the major reasons that our

approach is not as highly accurate as the approach of Kalogerakis et al. [2012] with respect to a fully automatic style transfer. We handle this limitation regarding the patches with the brushing interaction presented in Section 5.6. The described overfitting problem could be tackled by using more extensive training data, i. e., to learn an illustrator's hatching style from a multitude of illustrations of different objects. The preparation of the segmentation image and the registered 3D model, however, is quite labor-intensive. This requirement for a manual creation of prerequisites of the learning procedures is another drawback of our method.

Furthermore, our stroke rendering method exhibits certain problems. First, our textured strokes appear to be cut off at the tips because of the following problem. We use the trajectories of the strokes detected in the example image as a basis for both the widths and the texture coordinates of the rendered strokes. These trajectories are based on the skeleton of the regions formed by the strokes. The skeleton does not traverse the entire extent of a stroke region but starts and ends with an offset from the actual tip and end of a stroke. Using these locations to sample the stroke widths and stroke shape results in rendering strokes which are cut off at the tips. This leads to visual discontinuities at the borders of hatching patches. Multiple parallel strokes cut off at similar locations can form unwanted hard edges at the patch borders. We reduce this negative effect by adding an additional control point at the tip and end of an example stroke. Second, for some strokes it is inevitable to erroneously sample black color from a neighboring stroke or from the contour, resulting in unwanted artifacts. To avoid this, we simply omit these strokes for rendering. This omitting unfortunately implies that we need to leave away some example stroke patches which lead to severe artifacts. Third, distortion effects can appear when an example stroke is used for texturing a stroke whose trajectory strongly deviates from the trajectory of the example stroke. Our stroke rendering, therefore, does not always yield satisfactory quality. Better results could be achieved with extracting a set of representative stroke textures from the example illustration, where each texture contains one separate stroke.

A limitation of our image processing procedure is that it is restricted to example images with separated individual strokes. It fails in detecting the strokes in hatchings with many overlapping strokes. More elaborate image analysis would be necessary to detect the strokes in such imagery.

Another limitation of our approach is that both the speed of the automatic style transfer as well as the interaction granularity depend on the mesh resolution. The performance of inferring hatching patches and directions at the vertices is linearly proportional to the number of vertices of the target mesh. We thus have to work with low-resolution meshes to facilitate a reasonably fast inference of these two properties when using our system for real-time animation purposes. Using a mesh

of 20k triangles, we can infer patch labels and stroke directions at 3 fps on an Intel Core 2 Duo machine with 3GB RAM. This low polygon count naturally affects the quality of the result with respect to depicting surface detail. A benefit of hatching low-polygon models, however, is that the various interpolations during the generation of hatching strokes result in smooth strokes that create an impression of smooth shapes for rather blocky meshes. While the speed of the fully automatic synthesis benefits from a low mesh resolution, the brushing interactions benefit from a high mesh resolution. A higher resolution enables the user to brush hatching patches in more detail and to adjust hatching directions with a finer granularity. This fine-grained control improves the users' creative freedom and editing possibilities. In an interactive setting, a higher mesh resolution is thus desirable, and we here worked with mesh resolutions ranging from 35k to 90k triangles. This higher mesh resolution does not hinder a responsive editing of hatching patches and hatching directions because the global automatic inference of hatching patches is turned off in this interactive setting and because the stroke directions are only inferred at the currently edited vertices.

In the process of developing our approach, we learned that we can automatically transfer local characteristics, such as varying distances along a stroke or the local stroke directions, more reliably than global characteristics, such as regions where to place strokes. While experimenting with different ways of capturing the local characteristics, we found out that simple measuring and re-applying the measured values was not sufficient and that we need machine learning techniques to capture these properties. We also learned about the granularity at which specific characteristics can be transferred and which data is required. For example, we first measured the stroke directions only locally at each reprojected stroke control point and tried to capture it only as the measured angle in which it deviates from the curvature direction. We learned that this strategy was not robust enough. First, it was not robust enough because the granularity was too fine: we could not robustly map the direction measured at only one location on the surface to another location. Second, it was not robust enough because the data involved was too little: we could not robustly map the deviation from the curvature at one location to the deviation at another location using only the curvature direction. For these reasons, we now transfer the stroke directions on a coarser granularity (as a patch-wise direction field) and involve more data (all surface features that we also use for the stroke regions). We had similar insights regarding the stroke distances, resulting in us now transferring the stroke distances on a stroke level.

5.9 CONCLUSIONS AND FUTURE WORK

In summary, we propose a novel approach for the interactive example-based generation of pen-and-ink hatching illustrations from 3D meshes. We present a new learning setup that makes it possible to learn the depiction style of a hand-drawn example image, including a way to infer 3D information related to the 2D example image. We propose an analytical representation of hatching patches and hatching strokes. This representation of drawing elements is coupled with an hierarchical style transfer model that captures rendering properties on four levels of abstraction. We introduce adaptive surface patches that incorporate global drawing characteristics and which can be used for the creation of hatching strokes in object space. We present ways to capture and reproduce the directional characteristics on a patch level and the distance characteristics of hatching strokes locally on a stroke level. Finally, we provide novel interaction methods that allow users to directly and intuitively modify the resulting illustration. This interaction capability improves upon the static nature of previous methods.

We can reproduce some of the characteristics of the example illustrations. Our method can transfer directional characteristics with the help of an example-based patch-wise stroke direction field. Our method can also reproduce patterns of patch-wise and local stroke distance relationships, which results in hatching strokes that are less uniform and regular than the hatching strokes generated by existing methods. The transfer of global characteristics incorporated in the surface patches has some limitations. Therefore, our method does not produce as highly accurate results as the method of Kalogerakis et al. [2012] with respect to a fully automatic style transfer. We propose possible ways of improving upon this limitation. We also provide a brushing interaction that copes with this problem and that gives direct and intuitive control over hatching regions to the users of our system. Together, the proposed methods allow us to generate computer hatchings that arguably exhibit more 'character' than the results of previous techniques.

So far, we judge upon the quality of the results only by subjective reasoning. It would require more extensive evaluation to be able to judge upon the results less subjectively. One could envision a statistical assessment of the hatching properties as done for stippling by example [Kim et al., 2009]. But it would be even more interesting to conduct a field experiment designed as sort of a visual Turing test as proposed by Salesin [2002], reviewed by Gooch et al. [2010], and performed to a certain extent by Isenberg et al. [2006]. Showing the participants a set of hand-drawn and computer-generated illustrations, one would ask the participants whether the images were drawn by hand or generated by an algorithm. In this way one could examine how successfully the

hand-drawing characteristics can be reproduced, i. e., how well the human rendering process can be simulated, although the validity of such a visual Turing test is debatable [Isenberg, 2012; Hall and Lehmann, 2012]. Its validity is particularly debatable in our case because of the human intervention that happens in the image synthesis. Our results are not a sole product of computer generation due to this human intervention, which spoils the validity of a visual Turing test even more.

It would be interesting for future work to improve upon the described shortcomings of our style transfer model. With an enhanced style transfer, the visual appearance of the examples can be simulated even more closely. An improved automatic style transfer can further enhance the aesthetic quality of the results. An improved model would also allow for capturing a wider variety of hatching styles. We outline possibilities for improvement of our style transfer model in Section 5.8.

The possibility to brush with patches of hatching strokes onto a 3D model is a novel way of interacting with hatching renderings. It allows users to directly specify the desired hatching regions. The brushing interaction is very effective and pleasant to work with. The possibility for adjusting the illustration contrasts the static results of many comparable techniques. This creative freedom offered by our semi-automatic example-based approach makes it more likely that it is employed in a creative environment than it is likely for a fully automatic and static approach. We believe the proposed brushing interaction has the potential to serve as the basis for a tool for artists and illustrators. For future work, it would be interesting to further explore the interactive creation of hatching illustrations. The brushing of hatching patches can be extended with brushing metaphors to modify stroke properties such as distance, randomness, width, shape, etc. High-level brushes can be used for accentuation and abstraction as well as for modifying material properties. Integrating a layer support for multiple layers of hatching strokes can allow users to achieve various hatching effects. A design gallery showing previews for the result of using different example stroke patches can assist users to conveniently select the type of hatching patch to brush with.

Furthermore, we believe that the notion of explicitly represented dynamic patches embedded on the surface can be generalized to other illustrative rendering styles. Many methods for stylized rendering create such patches *implicitly* in image space, e. g., regions of homogeneous shading in cartoon rendering. An *explicit* patch representation, however, provides control over the type and location of patches. This control can be used by automatic methods, such as our automatic prediction of patches. And the control can also be employed for realizing user interactions with the illustration, such as our brushing tools. We are convinced that the notion of adaptive surface patches has great potential for future developments within illustrative rendering.

We discuss more ideas for future work as well as present some thoughts on potential applications of our interactive example-based hatching method in more detail in Chapter 6.

5.10 ACKNOWLEDGMENTS

We would like to thank the artists and illustrators that provided valuable feedback on our project. We also thank the Aim@Shape, VAKHUN, Google, and Polhemus repositories as well as the Princeton Graphics Group for the 3D models we used in this work. We also thank the developers of dlib, CGAL, and VolumeShop for their software.

This chapter is based on a previously published article [Gerl and Isenberg, 2013]. Moritz Gerl and Tobias Isenberg. Interactive Example-Based Hatching. In *Computers & Graphics*, 2013. In press.

6 CONCLUSION & FUTURE WORK

In this thesis we discuss two distinct methods for illustrative rendering. The two methods work with different input data, generate different rendering styles, and use different means of interaction. The two distinct methods are linked by being situated in the field of illustrative rendering and by being dedicated to large parts to the problem of implementing a useful and effective control over the result. We explore two different strategies for realizing control over the result in this thesis. In this chapter, we summarize the contributions of this thesis, draw conclusions from our most important findings, and present ideas for future work. Let us begin with concluding on Chapter 3.

6.1 SEMANTICS BY ANALOGY

The interactive semantics-driven volume visualization framework described in Chapter 3 improves upon the flexibility and usability of the semantic-layers approach by Rautek et al. [2007, 2008a]. It facilitates the injection of rule-based rendering functionality into arbitrary shader programs. This opens up the possibility to realize semantics-driven mappings based on arbitrary inputs and outputs with minimal implementation overhead. Furthermore, the graphical rule specification interface improves upon the shortcomings of a rule specification in natural language and permits users to intuitively and quickly explore different rule-based mappings assisted by visual feedback. An evaluation with two different potential target user groups indicated that the proposed concepts of parameter specification might be useful and well applicable in the examined target areas of radiology, neuroscience, and medical illustration.

In conclusion, the graphical approach for defining semantic mappings seems promising to us. We think that our graphical approach makes the power of semantics-driven volume rendering more readily accessible to the user than a textual rule formulation. Our graphical user interface also extends the original semantic-layers method with the capability to *explore* different semantic mappings. The evaluation with domain experts and medical illustrators confirmed that the graphical and brushing-centered interface is indeed promising for future development. We learned that using visual feedback in the form of previews and design galleries to assist the user in the parameter specification is beneficial for the usability and efficiency of a visualization system. Furthermore, the creative freedom

and possibility for local adjustments given by the brushing metaphor that we use for interaction is particularly interesting for artists and illustrators. The concepts that we propose in Chapter 3 can be improved and extended to create a direct volume illustration system suitable for artists and illustrators. The proposed concepts could, thus, help bridging the gap between illustrative rendering and fine arts, resp. between computer-generated and computer-assisted illustration.

Furthermore, it was interesting and informative to develop the concept of semantic shader augmentation. First, an automatic shader code generation during run-time was fascinating for us, as we are used to offline development with long compile times. Thus, we found it intriguing to modify a shader program and to directly see the impact of the modification without having to recompile and restart the framework software (VolumeShop [Bruckner and Gröller, 2005]). Note that the modification of the input shader results in updating both the graphical user interface as well as the result. Second, the possibility to realize a visualization mapping on arbitrary inputs and outputs was both fascinating and challenging for us. In the process of experimenting with various input properties for visualization rules (e. g., *normal, diffuse illumination, etc.*), we learned much about the behavior and usefulness of different data properties and data derivatives for rendering purposes. This learning process also informed the selection of surface features that we use as input to the rendering function in our hatching method (Chapter 5).

An interesting direction for future research in interactive semantics-driven volume rendering is to apply machine learning to automatically learn meaningful visualization mappings. Given a target output illustration style and a large set of input properties, one can apply machine learning to learn which input properties, parameters, and weights are decisive for achieving the target illustration style. The corresponding visualization rules can then be inferred automatically. This learned mapping can be provided to the user to be applied interactively without requiring the user to specify the input parameters and visualization rules that result in the desired illustration style. This user assistance by machine learning is similar to the style transfer via machine learning that we presented in Chapter 5. We think that such an automatic user-assistance would improve upon our semantics-by-analogy approach due to the following experiences with the interactive and flexible system. The concept of arbitrary inputs and outputs is so flexible that it was difficult for us to devise suitable mappings. We learned that the flexibility offered by the semantics-by-analogy approach offers a wide range of illustration possibilities, but is quite challenging to make use of in a meaningful way. Even for visualization experts it is not straightforward to come up with input parameters that are both intuitive to use and effective for illustration purposes. Drawing from our experiences gained in develop-

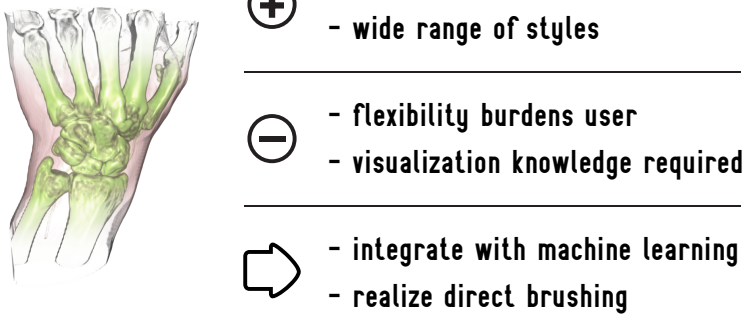


Figure 6.1: Conclusions on Chapter 3. An unextensive list of the benefits, disadvantages, and the most important ideas for improvement of the method in Chapter 3.

ing the example-based hatching approach described in Chapter 5, we are convinced that an integration of machine learning and interactive illustrative visualization would help alleviating the described drawback brought by the flexibility of the semantics-by-analogy approach. At the same time, the flexibility of the approach would still be made use of, only in a more automated manner.

The automatic inference of semantic mappings as outlined above can be complemented by replacing the brushing on intermediate data representations (i. e., the visualizations of semantic data properties) with brushing directly on the result image. In conclusion, we learned that brushing directly on the result is preferable to brushing in an intermediate domain. Our user evaluation also informed us that such a direct brushing would be useful and beneficial. Furthermore, the brushing of hatching patches developed in Chapter 5 confirmed our assumption that the direct application of a visual abstraction on the result is superior to the brushing on an intermediate domain. In the current form presented in Chapter 3, the intermediate domain is necessary because the data ranges intended to be included in the mapping have to be specified by the user. With the automatic learning of parameter configurations outlined here, however, this specification would no longer be necessary. The user could simply select a target rendering style and brush with this rendering style directly on the result.

To summarize our findings (see Fig. 6.1), we think that the approach presented in Chapter 3 is promising but would need to be further extended to be really applicable by the targeted user groups. Our main finding is that it would be beneficial to complement the interactive user control over the result by adding more automated control to the system. An integration with machine learning to automatically infer meaningful rules would improve upon the usability of the approach without limit-

ing its flexibility. Finally, brushing directly on the result would improve upon the directness of the approach and make it even more intuitive to use. This direct brushing behavior is similar to the brushing of hatching patches that we discussed in Chapter 5. Let us proceed with concluding on our hatching method.

6.2 INTERACTIVE EXAMPLE-BASED HATCHING

The fully interactive approach to controlling the result that we chose for the work presented in Chapter 3 is flexible but demands user interaction and an initial preparation of rendering parameters by a visualization expert. The example-based hatching method presented in Chapter 5, in contrast, was initially intended as a fully automatic method. In the process of developing the hatching method we learned, however, that our fully automatic approach in its presented form does not achieve the aesthetic quality at which we aimed. This quality issue is due to limitations of our style transfer model. The described limitation was the reason that we extended the fully automatic method with user interactions that allow to directly and locally adjust the generated hatching illustration. The resulting semi-automatic system enables user interaction but is still based on the automatic learning of a hatching style. This interactive example-based system did turn out to be quite effective in creating illustrative renderings that resemble hand-drawn examples. Let us recapitulate our explorations in implementing control over the result in illustrative rendering as the following. We explored a fully interactive illustration system in Chapter 3 and started out with a fully automatic system in Chapter 5. As a result of these explorations, we learned that the best solution for our goals was the '*golden mean*' between these two strategies in the form of an interactive example-based illustration system.

Let us briefly summarize the contributions of the hatching technique presented in Chapter 5. Our approach to hatching by example comprises the novelty of learning hatching styles from 'real-world' hand-drawn example illustrations. This is made possible by employing image processing to detect the strokes in the example and by using a 3D object registered with the example illustration. This setup allows us to include object-space measurements in learning the example drawing style. We further present a hierarchical style transfer model that learns and reproduces drawing characteristics in four different levels of abstraction (i. e., global, patch, stroke, and pixel level). We introduce the usage of adaptive surface patches as the basis for generating hatching strokes. These surface patches allow us to transfer global drawing properties and provide a robust control over hatching areas and the hatching properties applied in each individual area. Furthermore, we present an analytic representa-

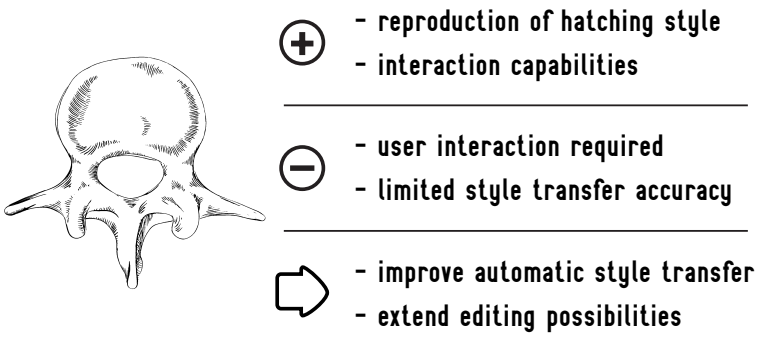


Figure 6.2: Conclusions on Chapter 5. An unextensive list of the benefits, disadvantages, and the most important ideas for improvement of the hatching method in Chapter 5.

tion of hatching strokes which, in combination with locally modeling the distances between hatching strokes along their extent, allows us to improve upon the regular hatching patterns generated by previous methods. The results presented in Chapter 5 demonstrate that our method can successfully transfer many characteristics of the example styles to a variety of target meshes. Furthermore, we complement the fully automatic style transfer of our approach with interaction capabilities. The key interaction of our approach is the brushing with hatching patches, which provides users with novel and effective possibilities to adjust the illustration according to their requirements and aesthetic judgment. Additional interactions allow users of our system to adjust the type, the general hatching direction, and the trajectories of the hatching strokes.

In conclusion (see Fig. 6.2), we learned that user interaction was the key to substantially increase the aesthetic and illustrative quality of the computer-generated hatching illustrations. The interaction possibilities presented in Chapter 5 are crucial for creating result images of the presented quality. At the same time, the automatic methods for realizing the hatching by example are just as important to achieve this visual quality. We learned that the combination of an automatic control over the result with an interactive control by the user does best meet our demands. The resulting system is a classical example for a semi-automatic system with high-level user control and low-level automated control. The user of our system can define the hatching regions (high level), while the computer automatically takes care of the details and generates the hatching strokes in the user-defined regions (low level). This semi-automatic approach distributes the control over the result in a way that allows for creating visually pleasing and effective illustrations. The high-level decisions that are most difficult to automate or are undesired to be automated, for example the decisions involved in determining the hatching regions, are put into the hands of the human. The low-level decisions and motor

controls that are most difficult to execute for the untrained human, for example the drawing of stroke trajectories that well describe the target surface, are put into the 'hands' of the machine. The lowest-level control in our hatching system currently is the fine-grained adjustment of stroke trajectories via retouching the reference stroke direction field. It would be desirable to extend the system with more low-level user interactions such as interactions for removing individual strokes or locally modifying the distances between neighboring strokes.

Furthermore, we learned that it is important to allow the user to work in various levels of detail. A flexible adjustment of the interaction granularity enhances the interaction efficiency and is necessary to adjust the user control to the current task. In our hatching system, this adjustment is particularly important for retouching the reference direction field to modify stroke trajectories. We allow the user to work in arbitrary levels of detail for this interaction. The level of detail can be adjusted by varying the brush size and by zooming. By making use of these adjustments the user can, e. g., rotate the reference field on the entire visible part of the surface with a single click, but can also zoom in and locally refine the trajectory of a singular hatching stroke. Up to now, we provide only a basic set of user interactions. It would be interesting to further exploit the potential of our hatching method in future developments.

The goal of the research presented in Chapter 5 was to improve upon the aesthetic quality of the results of existing hatching methods, which suffer from an overly regular and synthetic appearance. By our subjective judgment, we achieved this goal at least to a certain extent. We unfortunately had to discontinue the research on this project on hatching by example due to time and resource constraints. Up to this point, we have only partially tapped the potential of the described methods. Minor extensions to our technique in its current form can further improve the synthesis performance, the style transfer accuracy, the inter-frame coherence, the usability and editing possibilities, as well as the range of achievable rendering styles of our technique.

Synthesis performance: In the current form of our algorithms, the execution time spent on rendering the hatching strokes is negligible compared to the time needed for calculating the strokes. We give an example to illustrate the computational load of our method. A current bottleneck is the inference of the local stroke distance at each new stroke control point during the tracing of the stroke trajectories (see Fig. 5.14). This operation requires a projection to 2D, an intersection with the neighbor stroke, an evaluation of the stroke distance function (which implies an interpolation of the surface features), a spatial neighbor search to check if the new control point is too close to another stroke, as well as a reprojection to

3D. Note that these computations have to be performed for each new control point of every stroke. The proposed synthesis procedures can, however, be parallelized to some extent in order to increase the synthesis performance. A partial GPU implementation of our synthesis algorithms would most probably result in real-time frame rates.

Style transfer accuracy: The machine learning methods that we make use of can be replaced by other, more sophisticated learning methods. The parameters of the machine learning methods can be further optimized for our learning problem. Using a larger set of surface features would make the style transfer more robust, while the rendering performance issue would be alleviated by the GPU implementation described before. The overfitting problem can be tackled by using more extensive training data, i. e., by learning an illustration style from a multitude of illustrations instead of from only a single one. Together, these efforts would result in achieving a higher style transfer accuracy.

Temporal coherence: We did not work on achieving temporal coherence with our hatching method. The method in its current form cannot generate animations with temporal coherence. The proposed concepts of using adaptive surface patches as a basis for hatching strokes and of using the mesh geometry to trace stroke trajectories on the surface, however, are a solid basis for an inter-frame coherent hatching of 3D models. These analytic descriptions of drawing elements give a tangible control over the drawing elements, which is beneficial for achieving coherent animations. Further, the similarity of the hatching regions and hatching strokes between neighboring viewpoints can be exploited to achieve hatching animations with temporal coherence. One can reuse the hatching patches and strokes from the previous frame and adjust them to the view in the new frame, instead of re-generating the entire set of strokes for each frame. This would also further improve the rendering performance of our hatching technique.

Usability and editing possibilities: Our proposed user interactions can form the basis of an illustration system to be used by artists and illustrators. Our hatching method could as well form the basis for an illustration tool to be used by people without an artistic background. To realize these applications, however, the usability and the editing possibilities of our system would have to be extended. Our prototype system would have to be extended with functionality that is standard to graphics editing packages. This standard functionality includes, e. g., undo and editing history support, layer management, a graphical interface for selecting hatching style templates, etc. Furthermore, our basic set of user interactions can be extended with additional interactions. Brushing tools to

locally adjust stroke properties such as stroke width and distances would give additional low-level control. In addition to this, high-level brushing tools for accentuation and abstraction, for darkening and brightening, for adjusting material properties, etc. can be introduced.

Range of rendering styles: The concept of adaptive patches embedded on the surface can be extended and/or generalized to realize illustrative rendering styles other than hatching. For example, the surface patches can be combined with a texture-based NPR method for pencil or watercolor rendering. Regions or patches of similar visual attributes are inherently generated by many illustrative rendering methods (e. g., cartoon rendering creates images that consist of areas of one solid color). Representing the patches explicitly as we do in our hatching technique, however, has the great advantage that it provides full computational control over the location and shape of the patches, as well as control over the type and parameters of the visual abstraction that is generated within the patches. This control together with the possibility to transfer global style characteristics and the inter-frame stability of the adaptive surface patches in our opinion bears great potential for further developments within illustrative rendering.

Feedback learning: Finally, one can integrate a feedback loop in the learning procedure. A learned hatching style can be incrementally refined by re-running the learning algorithms on an illustration that has been adjusted with the proposed interaction tools. This feedback loop would make it possible to combine machine learning with human creativity in an even more powerful way than we do up to now.

6.3 ILLUSTRATIVE RENDERING & CONTROL OVER THE RESULT

Let us summarize our findings on automation and interaction in illustrative rendering. In general, we found that non-photorealistic or illustrative rendering methods can be enhanced by sophisticated means of implementing control over the result.

Providing the *user* with control over the result is particularly challenging in illustrative visualization because of the need of specifying the visibility of structures in addition to controlling the usually numerous NPR parameters. A commonly chosen approach in illustrative rendering of giving control over the result to the user is to provide users with a set of global rendering parameters. The adjustment of computer-generated illustrations by means of global parameters is, however, limited to a modification of the entire result and restricted in the range of visual effects that can be achieved. This restricts the creative freedom of the user and

prohibits an application of such NPR methods in creative environments. The described limitation can be dealt with by providing user interactions that allow users to directly, locally, and flexibly adjust the computer illustrations according to their requirements and aesthetic judgment.

Reserving the control over the result solely to the *computer* is the more traditional approach to implementing control over the result in illustrative rendering as compared to user interaction. Recent developments in these automatic methods focus on example-based approaches [Kim et al., 2009; Martín et al., 2011; Kalogerakis et al., 2012]. Such approaches perform an automatic learning of visual mappings from hand-crafted examples. These example-based approaches make it easier to use a multitude of inputs and outputs in visual mappings. As described in Chapter 2, the usage of multiple inputs and outputs allows to realize more sophisticated visual mappings, what improves upon the accuracy in which drawing, painting, or illustration styles can be simulated. In automatic approaches that do not perform any kind of parameter space analysis as well as in fully interactive approaches, the control of a multitude of input and output properties gets increasingly complex with an increasing number of inputs and outputs. The increasing complexity of the parameter space makes the parameter space more and more unintuitive to navigate for the programmer or the user. In example-based approaches, in contrast, the mapping of input properties to a rendering output is learned automatically. This automatic learning takes the burden of specifying adequate mappings away from the programmer or the user. The complex parameter space is then navigated automatically by the computer. This computer assistance enables the programmer or user to realize advanced illustrative visual mappings, which enhances the aesthetic and illustrative quality of the results. Apart from the pre-processing of the input image that is performed by Martín et al. [2011], previous example-based approaches for illustrative rendering are fully automatic. Fully automatic example-based approaches, however, lack the creative freedom and possibility for adjustments that is inherent to interactive approaches for illustrative rendering.

In the course of working on this thesis, we learned that the combination of the two different described approaches of implementing control over the result to an interactive example-based solution is most promising according to our experiences. This combination leverages the advantages of both an automated and an interactive control over the result. An interactive example-based approach allows for a twofold influence of human virtuosity on the rendering result. On the one hand, an example-based approach does capture and reproduce human virtuosity algorithmically. On the other hand, an interactive approach allows the human user to apply his or her virtuosity during the creation of the result.

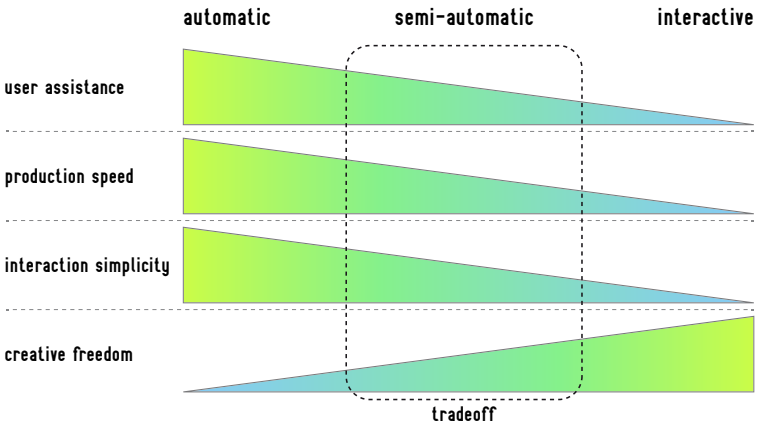


Figure 6.3: Tradeoff of advantages achieved by semi-automatic methods for illustrative rendering. The major advantage of interactive methods is the creative freedom they provide to the user (bottom row). This creative freedom is achieved, however, by sacrificing the advantages of automatic methods (top three rows). Semi-automatic methods partially leverage the advantages of both types of methods.

The proposed fusion of the two kinds of approaches for implementing control over the result to an interactive example-based system combines both of these influences. Such an interactive example-based system does improve upon the aesthetic and illustrative quality that is achievable with computer-generated illustrations. Furthermore, we believe that the described combination can help bridging the gap between illustrative rendering and fine arts, respectively between computer-generated and computer-assisted illustration. This bridging can help in finally *putting the artist in the loop* as proposed by Seims [1999], which would support the spreading of illustrative rendering methods in creative environments. Furthermore, the proposed notion of interactive example-based illustration systems can also inform the development of tools that allow people without an artistic background to interactively create visually appealing and effective renderings.

Furthermore, semi-automatic systems for illustrative rendering achieve a tradeoff of the advantages and disadvantages of automatic and interactive methods (see Fig. 6.3). The major benefit of fully interactive methods is the creative freedom that such methods provide to the user. This creative freedom is, however, achieved at the expense of sacrificing the benefits of fully automatic methods. These sacrificed benefits are properties such as user assistance, production speed, and interaction simplicity. Semi-automatic methods such as our hatching method include a tradeoff of the described properties and partially leverage the advantages of both automatic and interactive methods.

The combination of automatic and interactive control over the result allows non-experts in illustration to benefit from expert illustration knowledge and skills. The methods presented in this thesis allow people without an illustration background to create illustrations that they would otherwise not be able to create. By providing people without an illustration background hands-on access to illustration expertise we make illustrative rendering available in domains where it was not available before. Apart from this application by non-experts in illustration, our methods can also be applied by professional illustrators. We envision several potential application domains for the interactive illustrative rendering tools presented in this thesis. Fig. 6.4 shows an overview of potential application domains and some exemplary use cases of our methods. Let us conclude with outlining some of the potential applications.

Illustrators: One application domain for our methods is the application by professional illustrators, in particular by scientific and biomedical illustrators. Our methods can speed up the production process and therefore lower the expenses for producing scientific and medical illustrations. Furthermore, the employment of illustrative rendering methods by professional illustrators does harness the potential of the methods to a higher degree than the employment of such methods by computer scientists who develop the methods, as illustrators are skilled and trained in fine arts. We see two implications of this circumstance. First, the aesthetic quality and illustration effectiveness that is achieved with the illustrative rendering methods increases. Second, the exchange between the two disciplines of illustration and computer science fosters a mutual development that can result in even more sophisticated tools for computer-generated illustration. The interaction capabilities of the illustrative rendering methods that we present in this thesis make an employment of our methods by illustrators possible and feasible, as described in both Chapter 3 and Chapter 5. An employment of fully automatic methods by illustration experts would, in contrast, be highly unlikely due to the lack of possibilities for adjustment. We got this initial assumption confirmed from correspondence with professional illustrators.

Artists: Apart from the application for illustration purposes, the presented tools can also be applied for art creation. The result images depicted in this thesis are demonstrations of the technical capabilities of our methods and do have only little artistic value. Giving our tools in the hands of artists would result in tapping the creative potential of the tools and can lead to the creation of pieces of a novel art form.

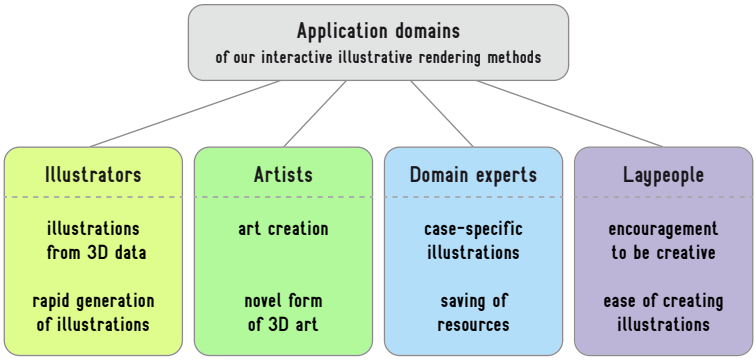


Figure 6.4: Application domains of our interactive illustrative rendering methods.

Domain experts: The previously described merit that our methods make illustration expertise accessible to non-illustrators facilitates an application of our methods by domain experts. Experts in various professions and disciplines can generate illustrations with the tools presented in this thesis instead of hiring professional illustrators. These can be illustrations for either publication or teaching purposes. The application of our methods by domain experts can increase the quality of the generated illustrations and allow domain experts to save the expenses for ordering professional illustrations. Let us back this claim up with the following comment of a professional medical illustrator on our work: '*I see a lot of bad and ugly illustration work, often made in powerpoint and then published in scientific publications. There will always be scientists that produce their own artwork instead of hiring an illustrator to do that for them. With an automatically generated image of this quality the visuals would be much better and thus improve the scientific publication.*'

Laypeople: Finally, the methods presented in this thesis can be applied by people without an artistic background, and other than domain experts, to interactively create illustrative renderings. One may argue that the prototype implementations of our methods are not yet applicable by laymen due to a lack of editing functionality. The proposed concepts can, however, inform the implementation of software products that are well applicable by laymen. This application by laypeople can serve two different purposes. The first purpose is to create illustrations to visually record and communicate ideas. The second purpose is to be creative and to generate art for the mere sake of pleasure and artistic expression. The interactivity and directness of our tools combined with the algorithmic image synthesis capabilities allow laypeople to create images and interactive renderings that they are otherwise not able to create. This computer assistance can stimulate the creativity of laypeople and increase their enjoyment of creating illustrations and art.

A

APPENDIX: SUPPLEMENTAL IMAGES

Now that we have discussed our methods we want to present some larger depictions of already shown results. We first show some enlarged versions of results of Chapter 3 and then of Chapter 5.

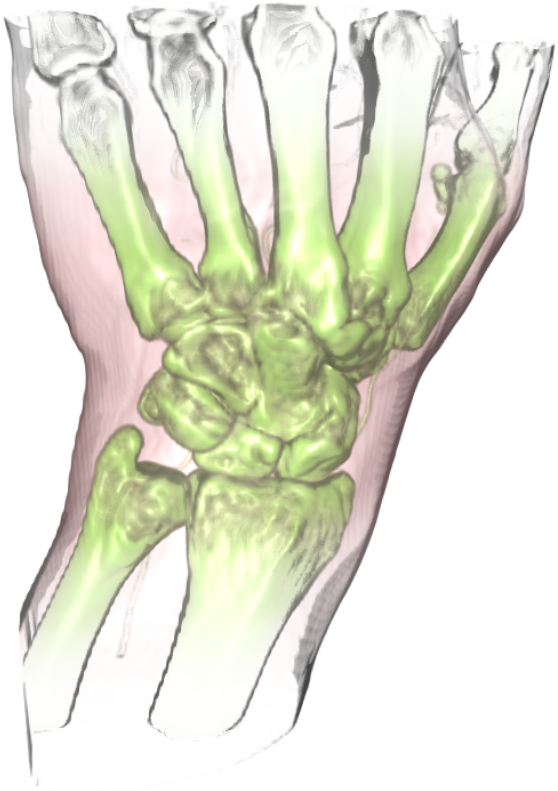


Figure A.5: Enlarged depiction of the hand visualization in Fig. 3.11g.

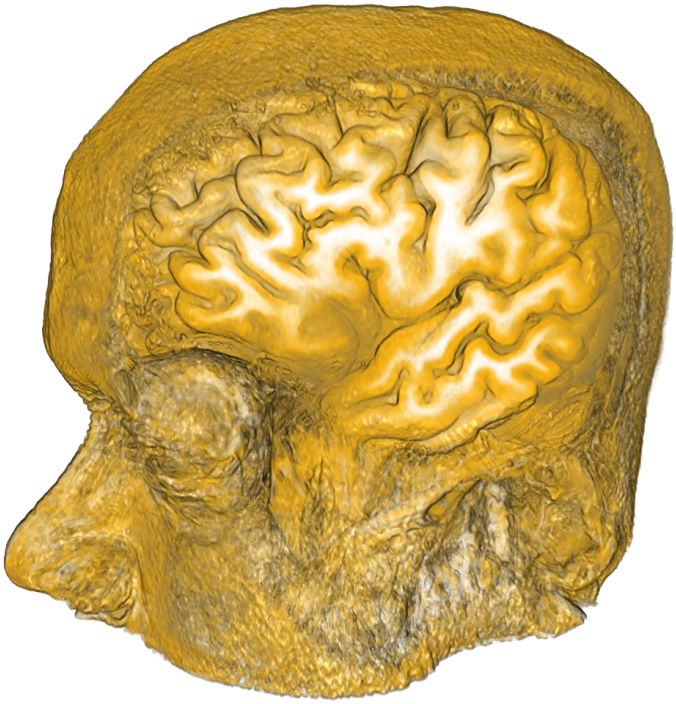


Figure A.6: Enlarged depiction of the brain visualization in Fig. 3.14d

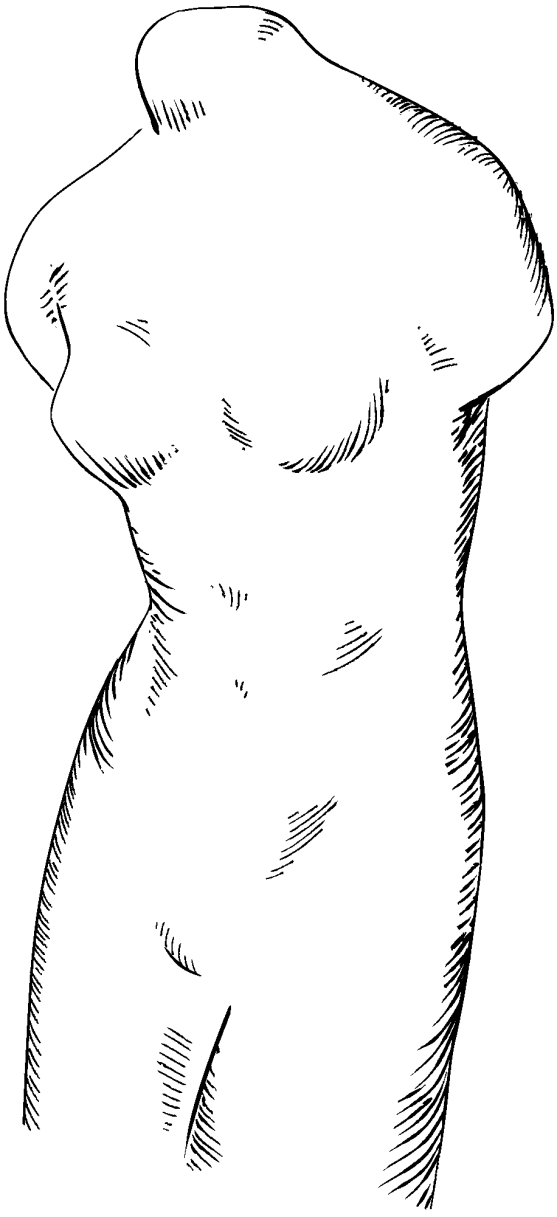


Figure A.7: Enlarged depiction of the venus illustration in Fig. 5.20.

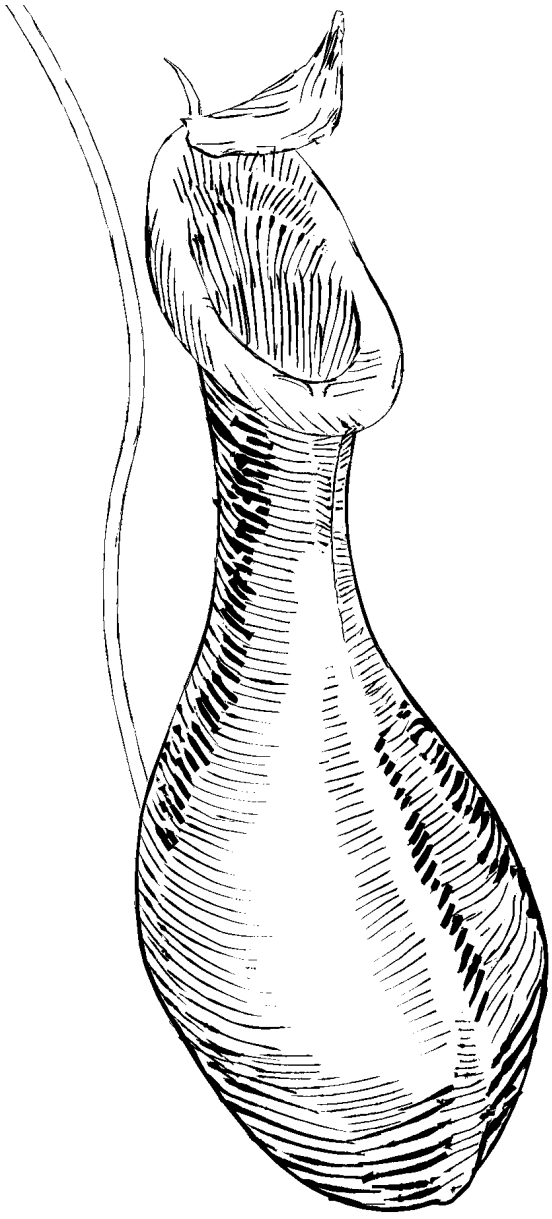


Figure A.8: Enlarged depiction of the pitcher plant illustration in Fig. 5.24a.

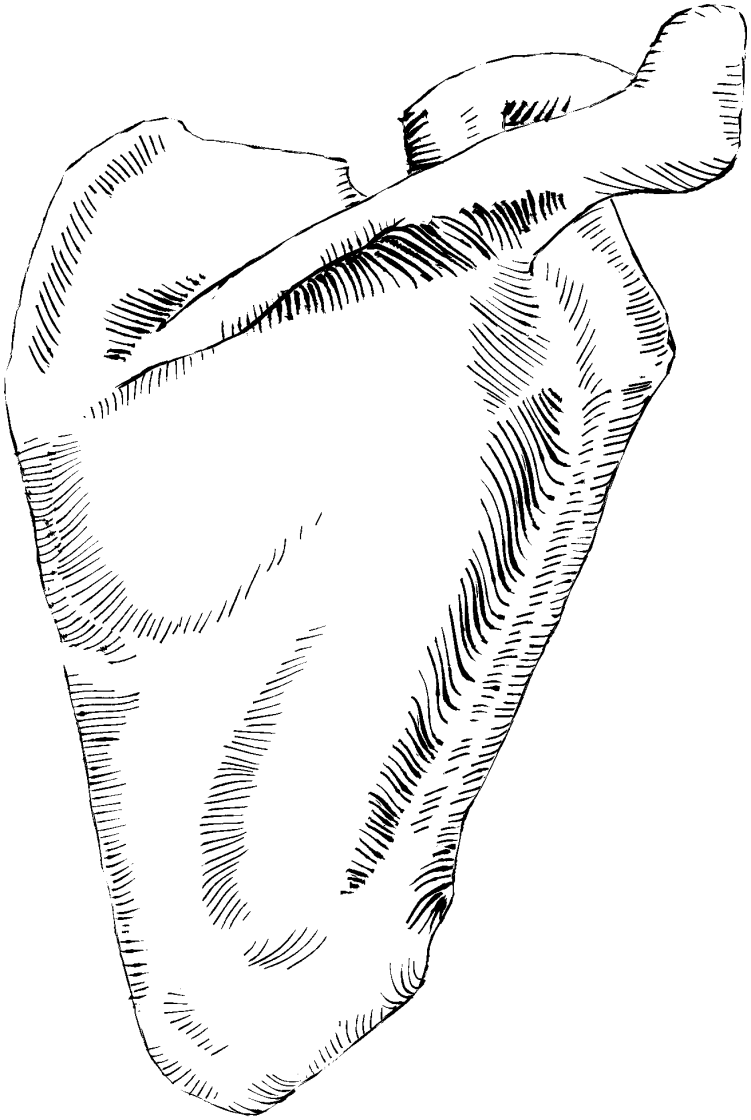


Figure A.9: Enlarged depiction of the scapula illustration in Fig. 5.24b.

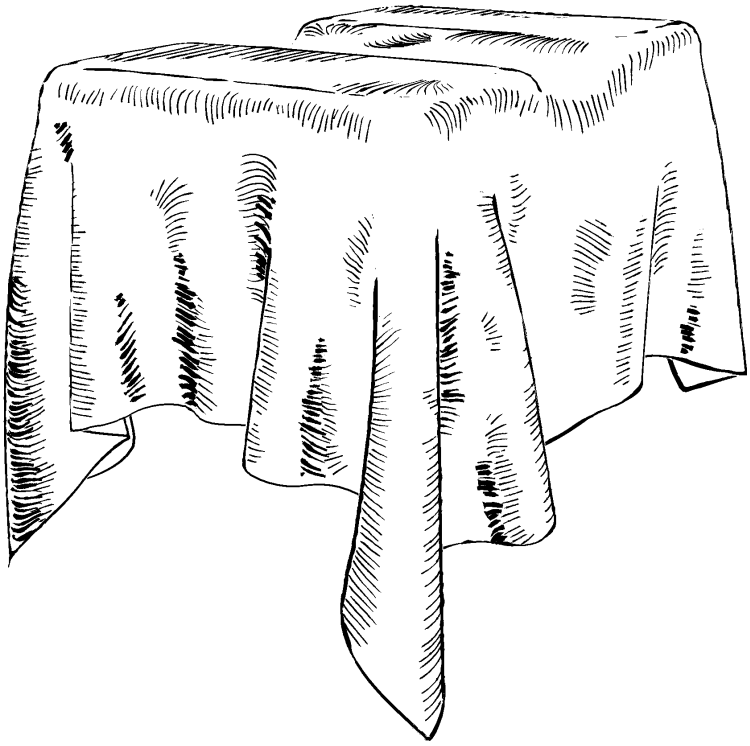


Figure A.10: Enlarged depiction of the two box cloth illustration in Fig. 5.27b.

B

APPENDIX: SUPPLEMENTAL VIDEOS

ACCOMPANYING the presented still images we also generated videos that demonstrate our approaches and that contain animations. Below we provide links to these videos as QR-codes and as URLs.

Figure B.11: QR-code that links to a video demonstrating the volume illustration method presented in Chapter 3.
URL: http://www.youtube.com/watch?v=Mq_qDqPjgQI

Figure B.12: QR-code that links to a video demonstrating the interactive hatching method presented in Chapter 5.
URL: http://www.youtube.com/watch?v=IuvUC3HFP44

BIBLIOGRAPHY

K.-i. Anjyo, S. Wemler, and W. Baxter. Tweakable Light and Shade for Cartoon Animation. In *Proc. NPAR*, pages 133–139, New York, 2006. ACM. doi> 10.1145/1124728.1124750

P. Barla, S. Breslav, J. Thollot, F. X. Sillion, and L. Markosian. Stroke Pattern Analysis and Synthesis. *Computer Graphics Forum*, 25(3): 663–671, September 2006. doi> 10.1111/j.1467-8659.2006.00986.x

P. Bénard, J. Lu, F. Cole, A. Finkelstein, and J. Thollot. Active Strokes: Coherent Line Stylization for Animated 3D Models. In *Proc. NPAR*, pages 37–46, Goslar, Germany, 2012. Eurographics Association. doi> 10.2312/pe/npar/npar12/037-046

S. Bischoff, T. Wey, and L. Kobbelt. Snakes on Triangle Meshes. In *Bildverarbeitung für die Medizin*, pages 208–212, Berlin/Heidelberg, 2005. Springer-Verlag. doi> 10.1007/3-540-26431-0_43

S. Breslav, K. Szerszen, L. Markosian, P. Barla, and J. Thollot. Dynamic 2D Patterns for Shading 3D Scenes. *ACM Transactions on Graphics*, 26(3):Article No. 20, July 2007. doi> 10.1145/1275808.1276402

S. Bruckner. *Interactive Illustrative Volume Visualization*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 3 2008.

S. Bruckner and M. E. Gröller. VolumeShop: An Interactive System for Direct Volume Illustration. In *Proc. Visualization*, pages 671–678, Los Alamitos, 2005. IEEE Computer Society. doi> 10.1109/vis.2005.135

S. Bruckner and M. E. Gröller. Style Transfer Functions for Illustrative Volume Rendering. *Computer Graphics Forum (Proceedings of Eurographics 2007, Prague, Czech Republic, September 3–7, 2007)*, 26(3): 715–724, September 2007. doi> 10.1111/j.1467-8659.2007.01095.x

S. Bruckner, S. Grimm, A. Kanitsar, and M. E. Gröller. Illustrative Context-Preserving Exploration of Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1559–1569, November 2006. doi> 10.1109/tvcg.2006.96

S. Carpendale. Evaluating Information Visualizations. In *Information Visualization: Human-Centered Issues and Perspectives*, volume

4950/2008 of *Lecture Notes in Computer Science*, pages 19–45. Springer-Verlag, Berlin/Heidelberg, 2008. doi> 10.1007/978-3-540-70956-5_2

X. Chen, A. Golovinskiy, and T. Funkhouser. A Benchmark for 3D Mesh Segmentation. *ACM Transactions on Graphics*, 28(3):Article No. 73, 2009. doi> 10.1145/1531326.1531379

F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewic. Where Do People Draw Lines? *ACM Transactions on Graphics*, 27(3):Article No. 88, August 2008. doi> 10.1145/1360612.1360687

B. Coyne and R. Sproat. WordsEye: An Automatic Text-to-Scene Conversion System. In *Proc. SIGGRAPH*, pages 487–496, New York, 2001. ACM. doi> 10.1145/383259.383316

B. Csébfalvi, L. Mroz, H. Hauser, A. König, and M. E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum*, 20(3):452–460, 2001.

W. Dauber, G. Spitzer, and H. Feneis. *Feneis' Bild-Lexikon der Anatomie*. Georg Thieme Verlag, 9th edition, 2005. ISBN 3-13-330109-8.

O. Deussen, S. Hiller, C. van Overveld, and T. Strothotte. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum*, 19(3):40–51, August 2000. doi> 10.1111/1467-8659.00396

D. Ebert and P. Rheingans. Volume Illustration: Non-Photorealistic Rendering of Volume Models. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 195–202. IEEE Computer Society Technical Committee on Computer Graphics, 2000. doi> 10.1109/VISUAL.2000.885694

G. Elber. Line Art Rendering via a Coverage of Isoparametric Curves. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):231–239, September 1995. doi> 10.1109/2945.466718

W. T. Freeman, J. B. Tenenbaum, and E. Pasztor. An Example-Based Approach to Style Translation for Line Drawings. Technical Report TR-99-11, MERL – A Mitsubishi Electric Research Laboratory, 1999.

W. T. Freeman, J. B. Tenenbaum, and E. C. Pasztor. Learning Style Translation for the Lines of a Drawing. *ACM Transactions on Graphics*, 22(1):Article No. 2, January 2003. doi> 10.1145/588272.588277

R. Fuchs, J. Waser, and M. E. Gröller. Visual Human+Machine Learning. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1327–1334, November/December 2009. doi> 10.1109/TVCG.2009.199

R. Gasteiger, C. Tietjen, A. Baer, and B. Preim. Curvature- and Model-Based Surface Hatching of Anatomical Structures Derived from Clinical Volume Datasets. In *Proc. of Smart Graphics*, volume 5166, pages 255–262, Berlin/Heidelberg, 2008. Springer-Verlag. doi> 10.1007/978-3-540-85412-8_25

R. Gasteiger, M. Neugebauer, O. Beuing, and B. Preim. The FLOWLENS: A Focus-and-Context Visualization Approach for Exploration of Blood Flow in Cerebral Aneurysms. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2183–2192, December 2011. doi> 10.1109/TVCG.2011.243

M. Gerl. Volume Hatching for Illustrative Visualization. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, November 2006.

M. Gerl and T. Isenberg. Interactive Example-Based Hatching. *Computers & Graphics*, 2013. In press. doi> 10.1016/J.CAG.2012.11.003

M. Gerl, P. Rautek, T. Isenberg, and M. E. Gröller. Semantics by Analogy for Illustrative Volume Visualization. *Computers & Graphics*, 36(2): 201–213, May 2012. doi> 10.1016/J.CAG.2011.10.006

A. Girshick, V. Interrante, S. Haker, and T. Lemoine. Line Direction Matters: An Argument for the Use of Principal Directions in 3D Line Drawings. In *Proc. NPAR*, pages 43–52, New York, 2000. ACM. doi> 10.1145/340916.340922

A. A. Gooch, J. Long, L. Ji, A. Estey, and B. S. Gooch. Viewing Progress in Non-Photorealistic Rendering Through Heinlein's Lens. In *Proc. NPAR*, pages 165–171, New York, 2010. ACM. doi> 10.1145/1809939.1809959

B. Gooch and A. A. Gooch. *Non-Photorealistic Rendering*. A K Peters, Ltd., Natick, 2001. doi> 10.1145/558817

S. Greenberg and B. Buxton. Usability Evaluation Considered Harmful (Some of the Time). In *Proc. SIGCHI*, pages 111–120, New York, 2008. ACM. doi> 10.1145/1357054.1357074

P. Hall and A.-S. Lehmann. Don't Measure—Appreciate! NPR Seen through the Prism of Art History. In P. Rosin and J. Collomosse, editors, *Image and Video based Artistic Stylisation*, chapter 16. Springer-Verlag, Berlin/Heidelberg, 2012. to appear.

J. Hamel and T. Strothotte. Capturing and Re-Using Rendition Styles for Non-Photorealistic Rendering. *Computer Graphics Forum*, 18(3): 173–182, September 1999. doi> 10.1111/1467-8659.00338

S. Hargreaves. Generating Shaders from HLSL Fragments. In W. Engel, editor, *ShaderX3: Advanced rendering with DirectX and OpenGL*, chapter 7.3, pages 555–568. 2005.

T. Hastie and R. Tibshirani. Classification by Pairwise Coupling. In *Proc. NIPS*, pages 507–513, Cambridge, MA, USA, 1998. MIT Press. doi> 10.1214/aos/1028144844

H. Hauser, L. Mroz, G.-I. Bischi, and M. E. Gröller. Two-Level Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, July–September 2001. doi> 10.1109/2945.942692

A. Hertzmann. A Survey of Stroke-Based Rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, July/August 2003. doi> 10.1109/mcg.2003.1210867

A. Hertzmann and D. Zorin. Illustrating Smooth Surfaces. In *Proc. SIGGRAPH*, pages 517–526, New York, 2000. ACM. doi> 10.1145/344779.345074

A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image Analogies. In *Proc. SIGGRAPH*, pages 327–340, New York, 2001. ACM. doi> 10.1145/383259.383295

A. Hertzmann, N. Oliver, B. Curles, and S. M. Seitz. Curve Analogies. In *Proc. EGWR*, pages 233–246, Goslar, Germany, 2002. Eurographics Association. doi> 10.1145/581896.581926

E. R. S. Hodges. *The Guild Handbook of Scientific Illustration*. John Wiley, 2nd edition, 2003. ISBN 9780471360117.

A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):69–82, February 1970. doi> 10.2307/1271436

T. Isenberg. Evaluating and Validating Non-Photorealistic and Illustrative Rendering. In P. Rosin and J. Collomosse, editors, *Image and Video based Artistic Stylisation*, chapter 15. Springer-Verlag, Berlin/Heidelberg, 2012. to appear.

T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, 23(4):28–37, July/August 2003. doi> 10.1109/mcg.2003.1210862

T. Isenberg, P. Neumann, S. Carpendale, M. C. Sousa, and J. A. Jorge. Non-Photorealistic Rendering in Context: An Observational Study. In *Proc. NPAR*, pages 115–126, New York, 2006. ACM. doi> 10.1145/1124728.1124747

B. Jobard and W. Lefer. Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proc. VisSci*, pages 45–55, Berlin/Heidelberg, 1997. Springer-Verlag.

P.-M. Jodoin, E. Epstein, M. Granger-Piché, and V. Ostromoukhov. Hatching by Example: a Statistical Approach. In *Proc. NPAR*, pages 29–36, New York, 2002. ACM. doi> 10.1145/508530.508536

R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In *Proc. SIGGRAPH*, pages 755–762, New York, July 2002. ACM. doi> 10.1145/566654.566648

E. Kalogerakis, A. Hertzmann, and K. Singh. Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics*, 29(4):Article No. 102, July 2010. doi> 10.1145/1778765.1778839

E. Kalogerakis, D. Nowrouzezahrai, S. Breslav, and A. Hertzmann. Learning Hatching for Pen-and-Ink Illustration of Surfaces. *ACM Transactions on Graphics*, 31(1):Article No. 1, 2012. doi> 10.1145/2077341.2077342

S. Kim, R. Maciejewski, T. Isenberg, W. M. Andrews, W. Chen, M. C. Sousa, and D. S. Ebert. Stippling by Example. In *Proc. NPAR*, pages 41–50, New York, 2009. ACM. doi> 10.1145/1572614.1572622

Y. Kim, J. Yu, X. Yu, and S. Lee. Line-art Illustration of Dynamic and Specular Surfaces. *ACM Transactions on Graphics*, 27(5):Article No. 156, December 2008. doi> 10.1145/1457515.1409109

D. E. King. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, July–September 2002. doi> 10.1109/TVCG.2002.1021579

J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg. State of the 'Art': A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*, 2012. To appear.

A. Lu, C. J. Morris, D. S. Ebert, P. Rheingans, and C. D. Hansen. Non-Photorealistic Volume Rendering Using Stippling Techniques. In R. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of IEEE Visualizaton 2002 (Boston, Massachusetts, October 2002*, pages 211–218, Piscataway, NJ, 2002. IEEE Press. doi> 10.1109/VISUAL.2002.1183777

E. B. Lum and K.-L. Ma. Lighting Transfer Functions Using Gradient Aligned Sampling. In *Proc. Visualization*, pages 289–296, Los Alamitos, 2004. IEEE Computer Society. doi> 10.1109/VISUAL.2004.64

E. B. Lum and K.-L. Ma. Expressive Line Selection by Example. *The Visual Computer*, 21(8–10):811–820, September 2005. doi> 10.1007/S00371-005-0342-Y

J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *Proc. SIGGRAPH*, pages 389–400, New York, 1997. ACM. doi> 10.1145/258734.258887

D. Martín, G. Arroyo, M. V. Luzón, and T. Isenberg. Scale-Dependent and Example-Based Stippling. *Computers & Graphics*, 35(1):160–174, February 2011. doi> 10.1016/J.CAG.2010.11.006

P. McCormick, J. Inman, J. Ahrens, C. Hansen, and G. Roth. Scout: A Hardware-Accelerated System for Quantitatively Driven Visualization and Analysis. In *Proc. Visualization*, pages 171–178, Los Alamitos, 2004. IEEE Computer Society. doi> 10.1109/VISUAL.2004.95

M. McGuire. The SuperShader. In W. Engel, editor, *Shader X4: Advanced Rendering Techniques*, chapter 8.1, pages 485–489. 2006.

T. Mertens, J. Kautz, J. Chen, P. Bekaert, and F. Durand. Texture Transfer using Geometry Correlation. In *Proc. EGSR*, pages 273–284, Goslar, Germany, 2006. Eurographics Association. doi> 10.2312/EGWR/EGSR06/273-284

M. Neugebauer, G. Janiga, O. Beuing, M. Skalej, and B. Preim. Anatomy-Guided Multi-Level Exploration of Blood Flow in Cerebral Aneurysms. *Computer Graphics Forum*, 30(3):1041–1050, June 2011. doi> 10.1111/J.1467-8659.2011.01953.X

M. Nijboer, M. Gerl, and T. Isenberg. Exploring Frame Gestures for Fluid Freehand Sketching. In E. Y.-L. Do and M. Alexa, editors, *Proc. SBIM*, pages 57–62, Goslar, Germany, 2010. Eurographics Association. doi> 10.2312/SBM/SBM10/057-062

V. Ostromoukhov. Digital Facial Engraving. In *Proc. SIGGRAPH*, pages 417–424, New York, 1999. ACM. doi> 10.1145/311535.311604

E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-Time Hatching. In *Proc. SIGGRAPH*, pages 581–586, New York, 2001. ACM. doi> 10.1145/383259.383328

P. Rautek, S. Bruckner, and M. E. Gröller. Semantic Layers for Illustrative Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1336–1343, November–December 2007. doi> 10.1109/TVCG.2007.70591

P. Rautek, S. Bruckner, and M. E. Gröller. Interaction-Dependent Semantics for Illustrative Volume Rendering. *Computer Graphics Forum*, 27 (3):847–854, May 2008a. doi> 10.1111/J.1467-8659.2008.01216.X

P. Rautek, S. Bruckner, M. E. Gröller, and I. Viola. Illustrative Visualization: New Technology or Useless Tautology? *ACM SIGGRAPH Computer Graphics*, 42(3):4:1–4:8, August 2008b. doi> 10.1145/1408626.1408633

C. Rezk-Salama, M. Keller, and P. Kohlmann. High-Level User Interfaces for Transfer Function Design with Semantics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1021–1028, September/October 2006. doi> 10.1109/TVCG.2006.148

P. Rheingans and D. Ebert. Volume Illustration: Nonphotorealistic Rendering of Volume Models. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):253–264, July–September 2001. doi> 10.1109/2945.942693

T. Ritschel, T. Grosch, and H.-P. Seidel. Approximating Dynamic Global Illumination in Image Space. In *Proc. I3D*, pages 75–82, New York, 2009. ACM. doi> 10.1145/1507149.1507161

P. Rosin and J. Collomosse, editors. *Image and Video based Artistic Stylisation*. Springer-Verlag, Berlin/Heidelberg, 2012. To appear.

C. Rössl and L. Kobbelt. Line Art Rendering of 3D-Models. In *Proc. Pacific Graphics*, pages 87–96, Los Alamitos, 2000. IEEE Computer Society. doi> 10.1109/PCCGA.2000.883890

F. Rössler, R. P. Botchen, and T. Ertl. Dynamic Shader Generation for GPU-Based Multi-Volume Ray Casting. *IEEE Computer Graphics and Applications*, 28(5):66–77, September/October 2008. doi> 10.1109/MCG.2008.96

T. Saito and T. Takahashi. Comprehensible Rendering of 3-D Shapes. In *Proc. SIGGRAPH*, pages 197–206, New York, 1990. ACM. doi> 10.1145/97880.97901

D. H. Salesin. Non-Photorealistic Animation & Rendering: 7 Grand Challenges. Keynote talk at NPAR, 2002.

M. P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin. Interactive Pen-and-Ink Illustration. In *Proc. SIGGRAPH*, pages 101–108, New York, 1994. ACM. doi> 10.1145/192161.192185

M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *Proc. SIGGRAPH*, pages 401–406, New York, 1997. ACM. doi> 10.1145/258734.258890

Y. Sato, C.-F. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S. Tamura, and R. Kikinis. Tissue Classification Based on 3D Local Intensity Structures for Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):160–180, April–June 2000. doi> 10.1109/2945.856997

J. Seims. Putting the Artist in the Loop. *ACM SIGGRAPH Computer Graphics*, 33(1):52–53, February 1999. doi> 10.1145/563666.563685

D. D. Seligmann and S. K. Feiner. Automated Generation of Intent-Based 3D Illustrations. *ACM SIGGRAPH Computer Graphics*, 25(4):123–132, July 1991. doi> 10.1145/127719.122732

A. Shamir. A Survey on Mesh Segmentation Techniques. *Computer Graphics Forum*, 27(6):1539–1556, 2008. doi> 10.1111/j.1467-8659.2007.01103.x

B. Shneiderman and C. Plaisant. Strategies for Evaluating Information Visualization Tools: Multi-Dimensional In-Depth Long-Term Case Studies. In *Proc. BELIV 2006*, New York, 2006. ACM. doi> 10.1145/1168149.1168158

P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, Berlin/Heidelberg, 2nd edition, 2003. ISBN 3540429883.

K. Stockinger, J. Shalf, K. Wu, and E. W. Bethel. Query-Driven Visualization of Large Data Sets. In *Proc. Visualization*, pages 167–174, Los Alamitos, 2005. IEEE Computer Society. doi> 10.1109/vis.2005.84

T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics. Modeling, Animation, and Rendering*. Morgan Kaufmann Publishers, San Francisco, 2002. doi> 10.1145/544522

N. Svakhine, D. S. Ebert, and D. Stredney. Illustration Motifs for Effective Medical Volume Illustration. *IEEE Computer Graphics and Applications*, 25(3):31–39, May/June 2005. doi> 10.1109/MCG.2005.60

C. Tietjen, R. Pfisterer, A. Baer, R. Gasteiger, and B. Preim. Hardware-Accelerated Illustrative Medical Surface Visualization with Extended Shading Maps. In *Proc. Smart Graphics*, pages 166–177, Berlin/Heidelberg, 2008. Springer-Verlag. doi> 10.1007/978-3-540-85412-8_15

M. E. Tipping and A. Faul. Fast Marginal Likelihood Maximisation for Sparse Bayesian Models. In *Proc. Artificial Intelligence and Statistics*, pages 3–6, Key West, Florida, 2003. Society for Artificial Intelligence and Statistics.

H. Todo, K.-i. Anjyo, W. Baxter, and T. Igarashi. Locally Controllable Stylized Shading. *ACM Transactions on Graphics*, 26(3):Article No. 17, July 2007. doi> 10.1145/1275808.1276399

F.-Y. Tzeng, E. B. Lum, and K.-L. Ma. An Intelligent System Approach to Higher-Dimensional Classification of Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, May/June 2005. doi> 10.1109/TVCG.2005.38

I. Viola, M. E. Gröller, M. Hadwiger, K. Bühler, B. Preim, M. C. Sousa, D. Ebert, and D. Stredney. Illustrative Visualization. In *Proc. Visualization*, number 5 in Tutorials. IEEE Computer Society, Los Alamitos, 2005. doi> 10.1109/VIS.2005.57

G. A. Winkenbach and D. H. Salesin. Computer-Generated Pen-and-Ink Illustration. In *Proc. SIGGRAPH*, pages 91–100, New York, 1994. ACM. doi> 10.1145/192161.192184

G. A. Winkenbach and D. H. Salesin. Rendering Parametric Surfaces in Pen and Ink. In *Proc. SIGGRAPH*, pages 469–476, New York, 1996. ACM. doi> 10.1145/237170.237287

J. Woodring and H.-W. Shen. Multi-Variate, Time Varying, and Comparative Visualization with Contextual Cues. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):909–916, September/October 2006. doi> 10.1109/TVCG.2006.164

S. Zachow, P. Muigg, T. Hildebrandt, H. Doleisch, and H.-C. Hege. Visual Exploration of Nasal Airflow. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1407–1414, November/December 2009. doi> 10.1109/TVCG.2009.198

J. Zander, T. Isenberg, S. Schlechtweg, and T. Strothotte. High Quality Hatching. *Computer Graphics Forum*, 23(3):421–430, September 2004. doi> 10.1111/j.1467-8659.2004.00773.x

M. Zhao and S.-C. Zhu. Portrait Painting using Active Templates. In *Proc. NPAR*, pages 117–124, New York, 2011. ACM. doi> 10.1145/2024676.2024696

LIST OF FIGURES

PUBLICATIONS RELATED TO THIS THESIS

JOURNAL PAPERS

Moritz Gerl, Peter Rautek, Tobias Isenberg, and Meister Eduard Gröller. Semantics by Analogy for Illustrative Volume Visualization. In *Computers & Graphics*, 36(2):201–213, 2012.

Moritz Gerl and Tobias Isenberg. Interactive Example-Based Hatching. In *Computers & Graphics*, 2013. In press.

PEER-REVIEWED CONFERENCE PAPER

Menno Nijboer, Moritz Gerl, and Tobias Isenberg: Exploring Frame Gestures for Fluid Freehand Sketching. In *Ellen Yi-Luen Do and Marc Alexa, eds., Proc. of the Sketch Based Interfaces and Modeling Symposium (SBIM 2010, June 7–10, 2010, Annecy, France)*. Goslar, Germany. Eurographics Association, pages 57–62, 2010.

PEER-REVIEWED CONFERENCE POSTER

Menno Nijboer, Moritz Gerl, and Tobias Isenberg: Interaction Concepts for Digital Concept Sketching. In *Holger Winnemöller and Marc Nienhaus, eds., Posters of the Seventh International Symposium on Non-Photorealistic Animation and Rendering* (NPAR 2009, August 1–2, New Orleans, USA). 2009. Extended abstract and poster.

NATIONAL SYMPOSIUM POSTER AND PAPER

Moritz Gerl and Tobias Isenberg: Image Analysis on Hatching Drawings. In *SIREN: Scientific ICT Research Event Netherlands*, (November 5, 2009, University of Twente, The Netherlands). 2009. Poster.

Menno Nijboer, Moritz Gerl, and Tobias Isenberg: Interaction Concepts for Fluid Freehand Sketching. In *Proc. of the Sixteenth Annual Conference of the Advanced School for Computing and Imaging* (ASCI Conference 2010, November 1–3, 2010, Veldhoven, The Netherlands). 2010. Paper number 12.

SAMENVATTING

Tegenwoordig veranderen illustratietechnieken steeds sneller. Dit proefschrift neemt deel aan dit proces van verandering en bestudeert de interactieve computergesteunde creatie van illustraties. Illustraties zijn effectieve middelen om informatie op een visuele manier vast te leggen en over te dragen. Het gebruik van illustraties reikt terug tot in de prehistorie, toen de prehistorische mens begon zijn omgeving af te beelden in vorm van grotschilderingen. Sinds toen heeft de kunst van de illustratie een lange evolutie doorlopen. Maar illustraties zijn nog steeds een belangrijke vorm van visuele communicatie. De ontwikkeling van illustratietechnieken is nauw verbonden met de ontwikkeling van nieuwe technologieën voor de beeldende vormgeving. Tegenwoordig zijn illustratietechnieken in een staat van verandering. Mensen gebruiken heden ten dage meer en meer digitale technieken voor het creëren van illustraties.

De digitale technieken die hierbij worden gebruikt zijn óf technieken voor de automatische beeldvorming óf technieken voor het interactieve digitale tekenen of schilderen. Beide deze soorten digitale illustratieve technieken zijn alleen toepasbaar voor specialisten in illustratie. Professionele illustrators zijn meestal mensen met een hoog artistiek talent, een opleiding tot beeldend kunstenaar, en vaak een speciale opleiding tot illustrator. Niet iedereen die een illustratie nodig heeft beschikt over de middelen om deze door een professionele illustrator te laten maken. Om deze reden zijn mensen die op zoek zijn naar een illustratie genoodzaakt om af te zien van de illustratie, om een bestaande illustratie te gebruiken die niet goed geschikt is, of om de illustratie zelf te creëren. Al deze oplossingen hebben een negatief effect op de kwaliteit van de illustratie en op het bereiken van het doel van de illustratie.

Dit proefschrift bestudeert technieken die expertise in illustratie beschikbaar en toepasbaar maken voor mensen zonder een creatieve achtergrond. Bovendien zijn de technieken die in dit proefschrift voorgesteld worden ook toepasbaar voor professionele illustrators. We behandelen in dit proefschrift technieken voor de interactieve illustratieve beeldvorming. Illustratieve of niet-fotorealistische beeldvorming is een deelgebied van de computergrafiek dat zich voornamelijk bezighoudt met het genereren van plaatjes in de stijl van met de hand gemaakte beelden. Een centraal oogmerk van de methoden in dit proefschrift is het verkrijgen van controle over de resultaten van illustratieve beeldvorming. Wij spreken hier van *controle* in de zin van *invloed* en *besturing* en niet in de zin

van *test* en *toetsing*. Methoden voor de illustratieve beeldvorming zijn meestal volledig automatisch, of stellen alleen beperkte mogelijkheden ter beschikking om de resultaten te controleren. Dit soort automatisering staat in contrast met de artistieke natuur van de plaatjes die gegenereerd worden. Bovendien belet het ontbreken van mogelijkheden voor ingrijpen, dat methoden voor illustratieve beeldvorming toegepast worden in creatieve omgevingen. De technieken die wij in dit proefschrift bestuderen kunnen helpen om deze toestand te verbeteren.

In hoofdstuk 2 onderzoeken we het onderwerp van controle over het resultaat in de illustratieve beeldvorming. Verder leggen we de samenhang uit van de twee verschillende methoden van illustratieve beeldvorming die in dit proefschrift worden voorgesteld.

In hoofdstuk 3 presenteren wij een methode voor de interactieve illustratieve visualisatie van volumetrische gegevens. Deze methode is gebaseerd op een techniek die het mogelijk maakt om visualisaties met behulp van regels te beïnvloeden. In dit proefschrift introduceren we een grafische methode om dit soort visualisatieregels te formuleren en te exploreren. Verder stellen we een mogelijkheid voor om de visualisatieregels lokaal aan te passen op een intuïtieve en grafische manier. Bovendien stellen wij een concept voor dat het mogelijk maakt om regel-gebaseerde visualisatiefunctionaliteit automatisch toe te voegen aan willekeurige shader-programmas. Dit is een erg flexibel concept dat het mogelijk maakt om willekeurige variabelen in shader-programmas als input en output voor visualisatieregels te gebruiken. De methode in hoofdstuk 3 maakt regel-gebaseerde visualisaties beter toegankelijk en toepasbaar voor de gebruiker en verbetert de mogelijkheden om dit soort visualisaties interactief te exploreren.

In hoofdstuk 4 gaan we dieper in op de inzichten met betrekking tot controle over het resultaat die we in hoofdstuk 3 hebben verworven. Gebaseerd op deze inzichten motiveren wij vervolgens de strategie voor controle over het resultaat voor hoofdstuk 5.

In hoofdstuk 5 introduceren wij een methode voor het interactieve genereren van plaatjes die op arceringen met pen en inkt lijken. Deze methode maakt het mogelijk om 3D modellen te renderen in de stijl van klassieke medische illustraties met pen en inkt. De techniek die wij in hoofdstuk 5 presenteren verbetert de esthetische kwaliteit van met de computer gegenereerde arceringen. Wij bereiken deze verbetering door de combinatie van twee strategieën voor controle over het resultaat: het automatische leren van voorbeelden en het interactieve ingrijpen door de gebruiker. Ten eerste stellen wij een voorbeeldgebaseerde methode van arcering voor. We maken gebruik van beeldverwerking en machine learning om de arceringsstijl van met de hand getekende illustraties te leren. Wij bespreken een model dat het mogelijk maakt om de geleerde arceringsstijl over te dragen op nieuwe 3D objecten. Dit model bevat

meerdere technische noviteiten op het gebied van voorbeeldgebaseerde illustratieve beeldvorming. Ten tweede presenteren wij interactietechnieken die de gebruiker van het arceringssysteem de mogelijkheid bieden om het resultaat te veranderen. De combinatie van automatische voorbeeldgebaseerde controle en interactieve controle over het resultaat maakt het mogelijk om de esthetische kwaliteit en de doeltreffendheid van met de computer gegenereerde arceringen te verbeteren. Bovendien maakt de techniek expertise in arcering toepasbaar voor mensen zonder kennissen in arcering. Daardoor woord het voor leken mogelijk gemaakt om illustraties te creëren die ze anders niet in staat zouden zijn te creëren. Verder maakt de interactiviteit van onze techniek het mogelijk dat hij gebruikt kan worden door kunstenaars en illustrators.

Tenslotte vatten we in hoofdstuk 6 de belangrijkste inzichten en technische bijdragen van dit proefschrift samen en trekken conclusies uit ons onderzoek. Bovendien presenteren wij ideeën en richtingen voor toekomstig werk. We sluiten dit proefschrift af met het omlijnen van mogelijke toepassingen van onze methoden.

ACKNOWLEDGMENTS

I am very grateful for being granted the privilege of conducting the research work presented in this thesis. I thank all the people who made this work possible, who contributed to my research, who supported me, and those who helped to accomplish my aims.

First of all, I thank my supervisor Dr. Tobias Isenberg for taking me as his PhD student and for his brilliant supervision. Dear Tobias, thank you for sharing your skills, knowledge, and wisdom. You helped me improve myself in many different ways. Thank you for investing that much time in my supervision and always being available for questions and advise. Thanks for your trust. Thank you for letting me develop my own ideas while giving competent advise in important strategic decisions as well as in sorting out all the tricky details of our techniques. It was a real pleasure to work with you. I feel honored by being your first PhD student.

Second of all, I thank my promotor Prof. Dr. Jos Roerdink for his calm and friendly leadership. Dear Jos, thank you for your guidance, support and advice. You taught me a lot about approaching problems with calmness, diligence, and consideration without loosing the touch to the eagerness brought by scientific curiosity and challenge. I also thank you for teaching me the importance of scientific rigor.

Furthermore, I thank Prof. Dr. Eduard Gröller for his advice and his support. Dear Meister, thank you for sharing your wisdom and expertise. I learned a lot from you about leadership and group dynamics. Thank you for offering me the great opportunity to collaborate with you and Peter Rautek. I also thank you, as well as Stefan Bruckner, for introducing me to the world of academic research during my master's thesis. VISIBLE FACIMVS QVOD CETERI NON POSSVNT.

Next, I very much thank the other members of my doctoral committee, Prof. Dr. Oliver Deussen and Prof Dr.-Ing. Bernhard Preim for their interest, time, and valuable comments on my thesis. I feel honored by having such brilliant people in my doctoral committee.

I also want to thank Dr. Peter Rautek for sharing his ideas and his knowledge. Dear Peter, thank you for the collaboration. It was great fun, very inspiring, and a real pleasure to work with you.

Furthermore, I thank the participants of our user studies for their time and for sharing their expertise. Many thanks to Peter van Ooijen from the UMCG radiology department and Aditya Hernowo from the BCN Neuroimaging Center. I also thank the artists and illustrators that gave me valuable advice and opinions on my projects. In particular, I

thank Avik Maitra, Sahal Merchant, Maartje Kunen, and Karin Spijker for their collaboration, interest, and feedback.

Next, I thank my fellow PhD student Lingyun Yu for being such a great person and officemate. Dear Yun, thank you for being there, for always having an open ear, and for your advice. It was a delight sharing the office with you. I wish you all the best.

I also thank my other fellow PhD students Wladimir van der Laan, Ronald van den Berg, Heorhiy Byelas, Bilkis Ferdosi, Alessandro Crippa, Maarten Everts, Ozan Ersoy, Deborah Mudali, Matthew van der Zwan, Jasper van de Gronde, and Andre Sobiecki for some interesting discussions and for creating a pleasant working atmosphere. The same goes to some of my fellow PhD students from other disciplines, Giulio Iacobelli, Ivan Vujacic, Hildeberto Jardon, and Ugo Moschini.

Many thanks to Esmee Elshof, Ineke Scheelhas, and Desiree Hansen for their administration, support, and help. Also thanks to Prof. Dr. Henk Bekker and Prof. Dr. Alexandru Telea for their comments and advice.

I thank all the people that I met, spent time with, and made friends with outside of the university during my stay in Groningen. Especially Julien, Oliver, Michael, Tino, and Jonas. Thank you for being my friends and for the good times together. All of you made it a very enjoyable time.

Also many thanks to my friends from home for supporting me and for not letting our friendship fade out despite the distance and little contact.

Thanks and kisses to Moni for your support and love during two years in which I worked on our relationship, myself, and this thesis.

Thanks to my brother Hannes for his advice and support. Thanks to Mika for being with Hannes. Also thanks to Markus for being there.

Finally, I thank my parents for making me, for raising me with so much love, and for supporting me all the time. Danke!

Moritz Gerl *Groningen, 23.09.2012*

COLOPHON

This thesis was typeset with LaTeX 2ε using Robert Slimbach's *Minion Pro* font. The typesetting is based on the *classicthesis* style by André Miede.